Niko Sandholm

Pilvipalveluiden kehittäminen

Uuden toiminnallisuuden toteuttaminen pilvipalveluun

Metropolia Ammattikorkeakoulu
Insinööri (AMK)
Tuotantotalous
Opinnäytetyö
3.4.2011

Metropolia

Tiivistelmä

Tekijä Otsikko Sivumäärä Aika	Niko Sandholm Pilvipalveluiden kehittäminen 59 sivua + 1 liite 3.4.2011
Tutkinto	Insinööri
Koulutusohjelma	Tuotantotalous
Suuntautumisvaihtoehto	Kansainvälinen ICT-liiketoiminta
Ohjaajat	Opettaja Thomas Rohweder Toimitusjohtaja Kimmo Strang

Insinöörityössä oli tavoitteena tuottaa uusi ominaisuus Genisys Oy:n olemassa olevaan CRM-pilvipalveluun. Uuden ominaisuuden avulla CRM-käyttäjien ja heidän asiakkaidensa välinen sähköpostiliikenne saadaan tallennettua palveluntarjoajan palvelimelle CRM-käyttäjän hyödynnettäväksi. Palvelun käyttäjien on mahdollista varastoida sähköpostiliikennettä myyjästä riippumattomasti sekä analysoida sähköpostiliikennettä paremman palvelukokemuksen tarjoamiseksi asiakkaillensa.

Ohjelmistoprojektin toteutuksen kannalta oli oleellista tutustua pilvipalveluiden ominaispiirteisiin. Havaittiin, että pilvipalvelut eroavat paikoittain perinteisistä ohjelmistotuotannon lopputuotteista mikä vaikuttaa oleellisesti kehityksessä käytettäviin menetelmiin. Käyttöliittymäsuunnittelu on oleellisessa osassa pilvipalvelun käyttäjälle tarjoamassa lisäarvossa ja voi olla ratkaiseva tekijä palvelun käyttökokemuksen kannalta.

Kehitysmenetelmäksi ohjelmistoprojektissa valittiin ketterä kehitysmenetelmä Scrum, koska se sopii ominaispiirteidensä vuoksi hyvin pilvipalveluiden kehitykseen. Työvälineinä kohdeyritys hyödyntää pääasiallisesti Java EE -kehitysalustaa muiden pilvipalveluidensa toteutuksessa, joten oli järkevää käyttää samaa ohjelmointikieltä mm. kielen tarjoamien mahdollisuuksien ja palveluiden yhdenmukaisen toteutuksen kannalta. Myös muita teknologioita hyödynnettiin kuten JavaScript-kieltä ja Ajax-tekniikkaa.

Ohjelmistoprojekti toteutettiin seuraamalla Scrum-menetelmän periaatteita mahdollisimman hyvin. Projekti aloitettiin tarkastelemalla palvelulle asetettuja vaatimuksia ja käytettäviä työvälineitä, minkä pohjalta hahmoteltiin projektisuunnitelma. Projekti koostui pääasiassa sovelluslogiikan ja käyttöliittymän rakentamisesta. Toteutuksessa oli vahvasti läsnä palvelu konseptin iterointi inkrementti kerrallaan kohti varsinaista lopputuotetta. Projektin päätyttyä uusi ominaisuus saatiin julkaistua yrityksen palvelimelle CRM-palvelun käyttäjien hyödynnettäväksi.

Avainsanat	pilvipalvelu, verkkopalvelu, CRM, käyttäjälähtöinen suunnittelu

Metropolia

Abstract

Author Title Number of Pages Date	Niko Sandholm Developing cloud services 59 pages + 1 appendix 3 May 2011
Degree	Bachelor of Engineering
Degree Program	Industrial Management
Specialisation option	Global ICT-business
Instructors	Thomas Rohweder, Lecturer Kimmo Strang, CEO

The objective for the thesis was to produce a new feature for the target company's CRM cloud service. The new feature would allow for the CRM-service users to get their email exchange between them and their clients forwarded to the service providers server to be stored and were it would be accessible for the CRM-user. This would allow for the CRM-service users to keep track on the email messages that have been sent by e.g. the company's selling personnel and use that information to provide better service experience to their customers.

For the software project to be successful it was necessary to get familiar with the characteristics of a web service. It was noticed that web services differ slightly from the outcome of traditional software development which will have an impact on the method that is used to produce the service. User interface design has great significance in the surplus which the web service provides for the end customer and can be crucial for the user experience obtained from the service.

An agile software development model Scrum was chosen for the project because of its characteristics that fit well with cloud services development. The service provider uses extensively Java EE development platform with its other cloud services so it was a natural choice with this project as well, due to the possibilities provided by the language and the coherency with provider's other services. So other technologies were also used, e.g. JavaScript-language and Ajax technique.

The software project was executed relying on the principles of Scrum as much as possible. The project was started by examining the requirements and tools to be used. And based upon those factors a design was made. The project consisted back-end and front-end parts of the service which needed to be build. In the execution phases the methods of agile development were actively taking place while the service was shaping into the final product. At the end of the project the web service was published to the CRM-users.

Keywords	Cloud service, CRM, user based design

Sisällys

1 Johdanto 2

1.1 Yrityskuvaus 2
1.2 Liiketoimintaongelma ja tavoite 2
1.3 Pilvipalvelut 4
1.3.1 Määritelmä 4
1.3.2 Palvelumallit 6
1.3.3 Perinteisestä ohjelmistotuotannosta pilvipalveluihin 7
1.3.4 Pilvipalveluiden merkitys yritykselle 10

2 Menetelmät ja työvälineet 12

2.1 Ohjelmistotuotanto 12
2.2 Pilvipalvelut ja käyttäjäkeskeisyys 15
2.3 Ketterä kehitys 20
2.3.1 Scrum 22
2.3.2 Ketterä sovelluskehitysprojekti käytännössä 25
2.4 Ohjelmointikielet 31
2.4.1 Java-ohjelmointikieli ja Java EE 5 -kehitysalusta 31
2.4.2 Web-teknologiat 35

3 Toteutus 40

3.1 Projektin alustus 40
3.2 Määrittely ja suunnittelu 41
3.3 Toteutus ja testaus 43
3.3.1 Sovelluslogiikka 44
3.3.2 Käyttöliittymä 47

3.4 Ylläpito 52

4 Johtopäätökset 54

4.1 Yhteenveto 54
4.2 Jatkotoimenpiteet 56
4.3 Projektin arviointi 56

Lähteet 58

Liitteet

Liite 1. Sovelluslogiikan ohjelmakoodi (vain yrityksen käyttöön)

Lyhenteet ja määritelmät

CRM	Customer Relationship Management
IaaS	Infrastructure as a Service
PaaS	Platform as a Service
SaaS	Software as a Service
XaaS	X as a Service
NIST	National Institute of Standards and Technology
ICT	Information and Communication Technology
W3C	World Wide Web Consortium
SLA	Service Legal Agreement
API	Application Programming Interface
Scrum	Ohjelmistokehitysmenetelmä
DAO	Data Access Object
Java	Ohjelmointikieli
Java ME	Java Micro Edition
Java SE	Java Standard Edition
Java EE	Java Enterprise Edition
XHTML	Extended Hypertext Markup Language
XML	Extended Markup Language
JavaScript	Ohjelmointikieli
AJAX	Asynchronous JavaScript and XML
DOM	Document Object Model
EIS	Enterprise Information System
JSP	JavaServer Pages
JSF	JavaServer Faces
POP	Post Office Protocol
IMAP	Internet Message Access Protocol
SQL	Structured Query Language

1 Johdanto

1.1 Yrityskuvaus

Genisys Oy on vuonna 2003 perustettu ohjelmisto- ja asiantuntijayritys, jonka päätoimintaa on asiakassuhdemarkkinointi. Yritys tarjoaa konsultointipalveluja markkinoinnin eri osa-alueilta, kuten asiakastietojen ja kohderyhmien hallintaa, asiakkuudenhallintaa ja digitaalisten markkinointi ratkaisuiden saralla. Yritys tarjoaa myös eri ohjelmistotuotteita pilvipalveluiden muodossa. Keskeisessä osassa ohjelmistopalveluissa ovat sähköpostimarkkinointiin suunniteltu GeniMail ja asiakkuudenhallintaan tarkoitettu GeniCRM. (Genisys 2010.)

GeniMail on selainpohjainen, markkinoinnin työkaluksi suunniteltu pilvipalvelu. Se tarjoaa valikoiman toiminnallisuutta, joiden avulla sähköpostit ja niiden lähetys saadaan optimoitua asiakkaiden hyödyksi. Palvelu tarjoaa viestien suunnittelua varten viestieditorin, sähköpostien lähetyksen ajastuksen optimaaliseen ajankohtaan ja mittareita lähetettyjen viestien analysointia varten. (Genisys 2010.)

GeniCRM on asiakkuuden hallintaan tarkoitettu palvelu. Se perustuu Saas-malliin ja on tarjolla asiakkaalle selainpohjaisena ratkaisuna. Palvelin sisältää mm. mahdollisuuden asiakastietojen, aktiviteettien eli asiakastapahtumien, myyntimahdollisuuksien ja projektien hallintaan. (Genisys 2010.)

Pilvipalvelut (ks. tarkemmin luku 1.3) ovat ohjelmistotuotteita, joita tarjotaan käyttäjille Internetin välityksellä. Genisys tarjoaa pilvipalveluita moduuleittain, jolloin käyttäjä voi vapaasti valita, mitä toiminnallisuutta haluaa käyttää, ja maksaa ainoastaan haluamastaan toiminnallisuudesta. Kirjauduttuaan palveluun käyttäjä pääsee navigoimaan haluamiinsa palveluihin, jotka käyttäjälle on aktivoitu. Kaikki palvelut ovat tarjolla saman käyttöliittymän alla, mikä mahdollistaa palveluiden luontevan, keskinäisen kommunikoinnin olettaen että käyttäjän käyttää useampaa Genisysin palvelua. (Genisys 2010.)

1.2 Liiketoimintaongelma ja tavoite

CRM- eli Customer Relationship Management -järjestelmien tarkoitus on avustaa ja palvella yrityksen myynti- ja markkinointiosastojen henkilöstöä, mm. toiminnan tehostamisen, palvelun laadun parantamisessa, ja pitkäkestoisten asiakassuhteiden

tehokkaassa hallinnassa. Olennaisten, ja yrityksen toiminnan kannalta pakollisten ominaisuuksien ja toiminnallisuuksien löytyminen CRM-järjestelmistä on avainasemassa järjestelmää markkinoivien yritysten, sekä palvelua käyttävien asiakasyritysten kannalta.

Yrityksen ja asiakkaan välillä käytävä dialogi on monimuotoinen, ja se tapahtuu useita eri tekniikoita käyttäen. Yrityksen ja asiakkaan välinen sähköpostiliikenne ja sen tutkiminen on oleellisessa osa tehokasta ja monipuolista asiakassuhteiden hallintaa. Analysoimalla ja monitoroimalla sähköpostiliikennettä yritys pyrkii muodostamaan syvällisemmän asiakassuhteen, erityisesti business-to-business -myyntiin suuntautuneilla toimialoilla, missä olemassa olevat asiakassuhteet ovat kriittinen osa jatkuvaa ja kasvavaa liiketoimintaa. Vaivalla hankitut asiakassuhteet pyritään säilyttämään, samalla kun etsitään uusia potentiaalisia asiakkaita.

Asiakastietokannat ovat yleisesti ottaen laajoja, ja kaikista asiakkaista pyritään pitämään mahdollisimman tarkkaa tietokantaa, jotta yrityksen kilpailukyky voidaan ylläpitää ja kasvattaa, ja palveluntasoa asiakaskohtaisesti parantamaan. Henkilöiden vaihtuvuus on myös otettava huomioon, ja kaikista asiakastapahtumista on oleellista voida pitää kirjaa. Näin ollen myös sähköpostiliikenne eri asiakkaiden välillä on syytä pystyä säilömään pitkäkestoisemman asiakassuhteiden hallinnan kannalta. Tämä myös mahdollistaa henkilöstöstä riippumattoman asiakassuhteiden hallinnan yritystasolla.

On oleellista sisällyttää CRM-järjestelmään ominaisuus ja toiminnallisuutta, joka mahdollistaa edellä määritellyn tarpeen. Yrityksen ja asiakkaan välinen sähköpostiliikenne pitää voida automaattisesti, tehokkaasti ja suoraviivaisesti rekisteröidä suoraan asiakastietokantaan, niin että tietoja on myös jälkikäteen helppo käsitellä. Tämä palvelee edellä kuvattua tarvetta asiakasyrityksen kilpailukyvyn kehittämiseksi.

Tässä työssä kyseessä oleva liiketoimintaongelman ratkaisemiseksi rakennetaan sähköpostien käsittely -ominaisuus ja integroidaan se olemassa olevaan GeniCRM-pilvipalveluun. Sähköpostin käsittely ominaisuuden avulla yritykset voivat lähettää sähköpostia asiakkaillensa, ja viestien sisältö ja mahdolliset liitetiedostot tallennetaan ohajustietojen avulla CRM-järjestelmään. Ominaisuus tuo lisäarvoa jo olemassa oleville CRM-asiakkaille helpottamalla heidän asiakassuhteidensa hallintaa ja edistää yrityksen CRM-palvelun haluttavuutta ja kiinnostavuutta myös uusien asiakkaiden silmissä.

1.3 Pilvipalvelut

Arkipäiväistyessään ja tullessaan yhä useamman käyttäjän ulottuville Internet tarjoaa uusia toimintamalleja ja mahdollisuuksia käyttäjille ja yrityksille. Yhteiskunnan muuttuessa tuotekeskeisemmästä toimintamallista palvelukeskeisempään suuntaan myös ohjelmistoteollisuus on alkanut kallistua erityisesti Internetin tehokkaaseen hyödyntämiseen uusia palveluita ja ansaintamalleja kehiteltäessä. Teknologioiden kehittyessä uudet palvelut voidaan rakentaa yhä helpommiksi, monipuolisemmiksi, tehokkaiksi ja luotettaviksi. Palvelumalli, jossain informaatio- ja kommunikaatioteknologiaa tarjotaan palveluna tuotteen sijaan, on ollut olemassa jo pidemmän aikaa. Nykyinen pilvipalvelumalli yhdistää moni olemassa olleita teknologioita ja malleja yhdeksi helpommin jäsennettäväksi kokonaisuudeksi. (Salo 2010: 7.)

1.3.1 Määritelmä

Pilvipalveluille ei ole olemassa virallista kansainvälistä määritelmää, jonka avulla niitä voitaisiin tarkasti luokitella ja selittää. Termille löytyy useita eri määritelmiä, jonka eri yritykset, ryhmät ja organisaatiot yms. instanssit ovat sille antaneet. Englanninkielinen termi cloud computing on kuitenkin ollut yleisesti käytössä ICT-alan yrityksissä ja -markkinoilla, ja monelle alalla toimivalle henkilölle termi on tuttu tai vähintäänkin suuntaa antava. Termi cloud computing oli Gartner tutkimustoimiston vuosittaisen strategisten teknologioiden listauksessa vuonna 2009 sijalla 3., ja vuonna 2010 se oli noussut 1. sijalle, mikä osittaa, että kyseessä oleva palvelumalli on tullut osaksi ICT-liiketoiminnan käsitteistöä. Pilvipalvelu ei itsessään ole mikään uusi innovaatio tai teknologia, vaan se on teknologioiden ja toimintaympäristön muutosten summa, joka mahdollistaa nykyään verkon yli toimivat palvelut. (Salo 2010: 9.)

Pilvipalvelu on hyvin itseään selittävä kielikuva. Yksinkertaistettuna pilvipalvelu viittaa pilvessä, eli Internetissä, sijaitsevaan tietotekniseen palveluun. Edellä mainittu on hyvin yleispätevä määritelmä, ja usein tilanteesta riippuen sitä on syytä jalostaa pidemmällekkin. Pilvipalvelu voi olla seuraavanlaisen esimerkin mukainen: Palveluntarjoaja tarjoaa asiakkailleen käytettäväksi palvelun ja/tai tarjoamansa tietotekniset resurssit verkon välityksellä, kaiken palvelun toiminnallisuuden ja resurssien fyysisesti sijaitessa palveluntarjoajan omissa tiloissa. Palveluntarjoaja siis vuokraa jalostettua tai jalostamatonta tietoteknistä toiminnallisuutta asiakkailleen. (Salo 2010: 16.)

Pilvipalveluilla on lukuisia ominaispiirteitä, joiden avulla niitä pyritään hahmottamaan. Hyvin keskeinen osa useimpia pilvipalveluita on se, että ne ovat luonteeltaan itsepalveluita eli käyttäjä itse on aktiivisesti mukana luomassa sisältöä palvelun tarjotessa dynaamisuutta ja toiminnallisuutta. Palveluntarjoaja antaa käyttäjille käytettäväksi tietyn toiminnallisuuden palvelumallista riippuen, mutta asiakkaan vastuulle jää palvelun hyödyntäminen. Käyttäjät myös saavat vapauden käyttää palveluaan joustavasti silloin kun heillä itsellään on siihen tarvetta, eikä käyttäjien tarvitse maksaa resursseista, jotka eivät ole tietyllä hetkellä käytössä. Palvelun käyttäjäksi on usein myös helppo liittyä ja irtaantua riippuen palvelun toiminnan luonteesta. (Salo 2010: 17-18.)

Palveluun on helppo päästä usein useammallakin erillisellä päätelaitteella, josta voidaan muodostaa Internet-yhteys. Pilvipalvelut ovatkin usein selainpohjaisia, mikä mahdollistaa yhden palvelutoiminnallisuuden rakentamisen usean eri alustoille suunnatun ohjelmisto version sijaan. Palvelua voi näin ollen käyttää myös tietokoneella, älypuhelimella, ja paikkariippumattomasti kotiverkosta tai työpaikan verkosta. (Salo 2010: 17-18.)

Palveluntarjoajan resurssit ovat kaikkien sen palveluita käyttävien käyttäjien käytössä. Kaikki asiakkaat ottavat päätelaitteellaan yhteyttä saman palveluntarjoajan laitteistoon omilla tunnustautumistiedoillaan ja voivat käyttää yhteisiä resursseja, jotka mukautuvat ja jakautuvat kaikkien käyttäjien kesken. Kunkin käyttäjän käyttämää resurssien määrää tietyllä ajanjaksolla pyritään mittaamaan tarkasti. Käyttäjiä laskutetaan palvelun käytöstä, joten pitää olla mahdollista tehdä tarkkaa raportointia siitä, miten asiakasta laskutetaan, ja toiminnan on oltava mahdollisimman läpinäkyvää. Yritys hyötyy raportoinnista myös kapasiteetin käyttöastetta silmällä pitäen. (Salo 2010: 17-18.)

Palvelun luonnetta voidaan havainnollistaa myös käyttäjäryhmittäin. Yhdysvaltalainen National Institute of Standards and Technology (NIST) esittää seuraavat vaihtoehdot

- yksityinen pilvi
- yhteisöllinen pilvi
- julkinen pilvi

- hybridi pilvi

Edellä mainitut määritelmät kuvaavat sitä, kuka palvelua ylläpitää ja kenen sitä on mahdollista käyttää. Yksityinen pilvi voi olla esimerkiksi yrityksen intranet, johon on rakennettu toiminnallisuutta ja palveluja, joita vain yrityksen työntekijät voivat käyttää. Yhteisöllinen pilvi on tietyn ryhmittymän tai useamman organisaation yhteiskäyttöön tarkoitettu pilvi. Julkinen pilvi on kaikille avoin palvelu, jonka ylläpidosta vastaa palveluntarjoaja. Hybridipilvi on yhdistelmä kaikki kolme edellä mainittua, esimerkiksi osan toiminnallisuudesta ollessa julkista ja osan yksityistä. (Salo 2010: 19.)

1.3.2 Palvelumallit

Pilvipalvelut voidaan jakaa kolmeen pääluokkaan, jotka jakautuvat lähinnä käytettävän arkkitehtuurin, resurssien ja asiakkaan haluamien/tarvitsemien vapauksien mukaan. Kolme päämallia ovat

- Infrastructure as a service (IaaS)
- Platform as a service (PaaS)
- Software as a service (SaaS).

Infrastruktuuri mahdollistaa alustan tarjoamisen palveluna, joka taas mahdollistaa sovellusten tarjoamisen palveluna. Asiakkaan vapaus on korkeimmillaan infrastruktuuri mallissa ja vähimmillään palvelumallissa. Asiakkaan vastuu taas korreloi vastakkaisesti olleen suurimmillaan infrastruktuurimallissa ja vähäisimmillään palvelumallissa. (Salo 2010: 22.)

Infrastruktuuri palveluna on fyysisen tietokonelaitteisto infrastruktuurin tarjoamista palveluna. Palveluntarjoaja antaa asiakkaalle käyttöön tietoliikenne infrastruktuurin ja asiakas voi yleensä käyttää palveluntarjoajan resursseja vapaasti ja joustavasti tiettyyn ennalta määriteltyyn pisteeseen asti, minkä jälkeen käyttäjä sopii palveluntarjoaja kanssa käyttäjän tarpeiden mukaisesta lisäkapasiteetista. Tässä mallissa asiakkaalla on eniten vapauksia, mutta myös vastuun määrä kasvaa, koska palveluntarjoaja tarjoaa fyysiset resurssit ja kattaa niihin liittyvät vastuukysymykset. Asiakkaan pitää itse ottaa monia eri asioita ja näkökulmia pitää ottaa huomioon, kuten laitteistolla olevien tiedostojen tietoturva, asiakkaan tarjoaman palvelun toimivuus jne. Palvelun on skaalautuu yleensä hyvin eri tarpeisiin ja mahdollistaa periaatteessa rajattoman

määrän resursseja kiinteällä kuukausi ja/tai käyttöön perustuvalla laskutuksella. (Salo 2010: 25-28.)

Kehitysalusta palveluna on toimintaympäristön eli palveluiden kehitysalustan tarjoamista palveluna. Siinä palveluntarjoaja antaa asiakkailleen käytettäväksi esimerkiksi kehitysympäristön, missä asiakkaat voivat luoda itse tuottaa palveluita omille loppukäyttäjilleen. Tässä mallissa vapaus ja vastuu ovat kiinni palveluntarjoajasta ja sen tarjoamasta sopimuksesta, mutta yleisesti sijoittuu IaaS- ja SaaS-mallien väliin. Palveluntarjoaja antaa asiakkaan käyttöön tietyn kehitysalustan ja tarjoaa siihen eri moduuleita. Esimerkki PaaS-mallista on Googlen tarjoama AppEngine, jolla voi tuottaa ohjelmistoja Java- ja Python-kielillä. (Salo 2010: 28-29.)

Ohjelmisto palveluna on tunnetuin palvelumallimuoto. Siinä asiakkaan päätäntävalta on vähäinen edellisiin malleihin verrattuna, mutta myös kaikki vastuu siirtyy palveluntarjoajalle, joka pitää huolen mm. palvelun toimivuudesta ja tietoturvasta. Asiakas vapauttaa omia henkilöstöresurssejaan ohjelmistojen ylläpidosta ja päivittämisestä, ja kertaluontoiset ohjelmistolisenssi yms. investoinnit muuttuvat juoksevaksi käyttöön perustuvaksi laskutukseksi. Palvelun mahdollistaa tiedon jakamisen kaikkein asiakkaan määrittelemien käyttäjien kesken, mutta jokainen käyttäjä saa kuitenkin yksilöllisen käyttäjäkokemuksen omilla tunnustautumistiedoillaan. Esimerkkejä Saas mallista ovat Google Apps -toimisto-ohjelmistojen kokoelma ja Elisan tarjoama OIWA -viestintäpalvelukokonaisuus. (Salo 2010: 29-30.)

1.3.3 Perinteisestä ohjelmistotuotannosta pilvipalveluihin

Ero perinteisen ohjelmistotuotannon ja pilvipalveluiden välillä on selkeä. Tuotekeskeisestä ohjelmistokehityksestä on siirrytty palvelukeskeiseen, mikä sisältää ajattelutavan, tiettyjen näkökulmien ja painopisteiden siirtymistä eri suuntaan. Syyt painopisteen siirtymiselle ovat moninaiset, kuten aiemmin on todettu. Teknologian kehittyminen on ollut suuressa osassa pilvipalveluiden kehityksessä. Teknologian kehittyminen pitää sisällään yleisen pilvipalveluiden käyttö infrastruktuurin eli Internetin palveluiden kehittämiseen vaadittavat työkalut ja niiden monipuolistuminen ja päätelaitteiden kehittymisen ja leviämisen yhä useamman käyttöön. Taustalla kehitystä ovat ajaneet yritysten tarpeet kustannussäästöihin sekä tuotannon nopeuttamiselle ja joustavuudelle on ollut tarve ICT-alan kehittyessä. (Salo 2010: 42-43.)

Tietoliikenne-infrastruktuuri on suuressa osassa pilvipalveluiden kehittymisessä ja yleistymisessä, ja infrastruktuuri on ainoa elementti, joka tällä hetkellä muodostaa lievän ns. pullonkaulan palveluiden kehittämisen kannalta. Pilvipalvelut vaativat yleistyäkseen toimivan ja varman tietoliikenneinfrastruktuurin, joka on yleisesti ihmisten ulottuvilla ja käytettävissä. Tietoliikenneinfrastruktuuria voidaan verrata sähkönjakeluinfrastruktuuriin. Ainoastaan luotettava ja kapasiteetiltaan tarpeeksi tehokas jakeluinfrastruktuuri mahdollistaa palvelun yleistymisen yrityskäytössä. Jos esimerkiksi tietosiirtonopeudet ovat kapasiteetiltaan liian pienet, palvelut toimivat hitaasti tai äärimmäisissä tapauksissa tiedonsiirtonopeutta joudutaan kompensoimaan palveluiden yksinkertaistamisella. (Salo 2010: 59-60.)

Internetissä on myös siirrytty perinteisten staattisten verkkosivujen sijaan käyttämään paljion myös dynaamista sisältöä. Web 2.0:n myötä kehittäjille on tullut uusi mielenkiintoisia työkaluja ja mahdollisuuksia käyttäjäystävällisempien, dynaamisten ja tehokkaiden palveluiden kehittämiseen. Uudet työkalut ja menetelmät ovat ensin innokkaimpien kehittäjäryhmien kokeilussa, ja jos teknologia näyttää tarpeeksi hyödylliseltä useamman alalla toimivan yrityksen mielestä, se saattaa pilkkuhiljaa ruveta hioutumaan standardiksi esimerkiksi World Wide Web Consortium (W3C) -organisaation toimesta. (Salo 2010: 54-59.)

Käyttöympäristö on muuttunut myös oleellisesti perinteiseen ohjelmistotuotantoon nähden. Käyttäjän ja palveluntarjoajan välillä on suhde palvelunkäyttöhetkellä, ja sekä asiakkaan että palveluntarjoajan resurssit ovat palvelunkehittäjien käytettävissä. Asiakkaan tarjoamat resurssit ovat kytköksissä Internet-selaimen tuomiin mahdollisuuksiin, ja palveluntarjoaja tarjoaa palvelimensa kapasiteetin verkkopalvelun muodossa. Useat W3C:n uudemmat standardit pyrkivät tarjoamaan selainriippumattomia rajapintoja, joiden avulla kehittäjät voivat luoda luotettavia palveluita, käyttäjän selainvalinnasta riippumatta. Perinteiset ohjelmointikielet, kuten esimerkiksi Java ja C++, mahdollistavat palveluiden logiikan ja toiminnallisuuden kehittämisen palveluntarjoajanpuolella. (Salo 2010: 54-59.)

Internet-selainten kehityksen myötä ohjelmoijat ovat voineet laittaa osan palveluiden toiminnallisuudesta toimimaan käyttäjän tietokoneen resurssien varassa laskien näin palveluntarjoajan serverien kuormitusta. JavaScript on tunnettu selainpuolella käytettävä ohjelmointikieli, ja sitä hyödynnetään nykyään laajasti pilvipalveluita

kehitettäessä. Se mahdollistaa interaktiivisuuden tuomisen verkkosivuille ja tehokkaamman verkkosivujen sisällön päivittämismahdollisuuden. (Salo 2010: 55-57.)

Laitteet, joilla ohjelmistoja käytetään, ovat myös muuttaneet muotoaan. Ohjelmistoja ja palveluita ei enää luoda ainoastaan pöytäkoneille, vaan päätelaitteet kattavat myös kannettavat tietokoneet, älypuhelimet ja kosketusnäytölliset tablet-laitteet. Eri alustojen määrän kasvaessa, laitteelle asennettavat ohjelmistot joudutaan räätälöimään laitteeseen sopiviksi, mikä tuottaa lisätyötä palveluntarjoajalle. Pilvipalveluiden kannalta laitteiden omat ominaispiirteet eivät ole ongelma. Jos päätelaitteessa on selain, tai mahdollisuus sen asentamiselle ja Internet-yhteys, myös pilvipalvelut usein toimivat. Tämä on suuri etu kehittäjien kannalta ja ohjelmistoja ja/tai palveluita tarjoavat yrityksen näkökulmasta, koska enemmän resursseja vapautuu itse palvelun toiminnallisuuden kehittämiseen. (Salo 2010: 61.)

Mobiilien päätelaitteiden pääongelmaksi voi muodostua käyttöpaikasta riippuen tietoliikenne-infrastruktuurin kehittymättömyys. Pilvipalvelua ei välttämättä ole mielekästä käyttää paikassa, jossa tietoliikennekustannukset kasvavat kohtuuttoman suuriksi palvelu käyttöön saavutettavaan hyötyyn nähden. Jos tietoliikenne-infrastruktuurin tarjoamat yhteysmahdollisuudet ovat kapasiteetiltaan vajavaiset, optimaalista palvelunkäyttökokemusta ei ole mahdollista saavuttaa. Vaikkakin langattomat yhteysmahdollisuudet ovat parantuneet ja kehittyneet suosi vuodelta, on mobiililaitteiden käyttö oleellisesti riippuvaisempi Internet-palveluntarjoajasta verrattuna kiinteää verkkoyhteyttä käyttävään laitteeseen.

Yrityksille tietotekniikkaresurssit, olivat ne sitten palvelimia tai ohjelmistoja, ovat olleet suuri yksittäinen kustannuserä, ja yritys maksaa jatkuvia juoksevia ylläpito- ja käyttökustannuksia, vaikka ne toimisivatkin osittain joutokäytöllä. Käyttämättöminä olevat tietotekniikkaresurssit ovat yritykselle huono investointi. Yrityksillä on tarvetta saavuttaa kustannussäästöjä ja maksimoida sijoitetun pääoman tuotto. Pilvipalvelumalli tarjoaa tähän tehokkaan ratkaisun. Pilvipalveluiden tarjoama joustavuus on omiaan tehostamaan yritysten toimintaa alati muuttuvilla markkinoilla. Mahdollisuus ottaa resursseja käyttöön ja poistaa niitä käytöstä suhteellisen yksinkertaisesti ja nopeasti auttavat yrityksiä sopeutumaan markkinoiden muuttuviin olosuhteisiin. Tulevaisuutta ja silloin tarvittavia resursseja on hankala ennustaa, mikä on oleellinen toimintariski

yrityksen näkökulmasta, varsinkin suuria pääomia vaativia hankintoja tehtäessä. (Salo 2010: 70-71.)

1.3.4 Pilvipalveluiden merkitys yritykselle

Nykyisten trendien valossa pilvipalvelu markkinat tulevat kasvamaan tulevaisuudessa käyttäjämäärien kasvaessa. Pilvipalveluissa on luonnollista potentiaalia, jolla yritys pystyy tehostamaan omaa liiketoimintaansa, ja palveluntarjoajat pystyvät kehittämään tehokkaampia palveluita yrityksille perinteisten ohjelmistotuotannon menetelmien pohjalta. Tapauskohtaista on, milloin yritys voi alkaa korvaamaan tietoliikennetoimintojaan pilvipalveluilla ja onko se kannattavaa ja mahdollista. (Salo 2010: 70.)

Perinteisesti hyötyjen ja haittojen arviointi alkaa yrityksen omien olemassa olevien IT-toimintojen tutkimisella: mikä on toimintojen tarjoama hyöty ja kokonaiskustannus? Näitä verrataan pilvipalvelun tarjoamiin vastaaviin hyötyihin ja haittoihin, ja pilvipalveluun liittyviin riskeihin ja niistä mahdollisesti seuraaviin kustannuksiin. Jos edellä mainitut vertausanalyysi osoittaa, että hyödyt ja säästöt ovat suhteellisesti tarpeeksi suuria ja riskejä voidaan minimoida ja hallita yritykselle siedettävällä riittävällä tarkkuudella. Silloin on syytä ruveta tutkimaan yrityksen sisäistä valmiutta uuteen pilvipalveluteknologiaan. (Salo 2010: 70-71.)

Yrityksen kulttuuri ja henkilöstön osaaminen ovat oleellisessa osassa pilvipalvelun hyödyn maksimoimisessa. Yritys, joka innovatiivisesti etsii uusia palveluita ja jossa henkilöstö on motivoitunutta kokeilemaan uusia toimintamalleja, pilvipalveluiden käyttöönotolle ei juuri asetu esteitä. Jos taas tilanne on päinvastainen, jolloin yrityksen ilmapiiri on vähemmän kannustava, pilvipalvelu saatetaan ottaa käyttöön ja todetaan pian hyödyttömäksi, koska kukaan ei sitä halua käyttää. Lainsäädäntö ja toimialan luonne voivat olla myös selkeä este. Jos tietoja luonteensa vuoksi liian salaista tai liiketoiminta vaatii 100 %:n toimintavarmuuden, pilvipalvelun käyttöönotto voi olla este. Tai yksinkertaisesti pilvipalvelu markkinoiden nuoruuden vuoksi tarjolla ei välttämättä vielä ole sellaista palvelua, johon yritys voisi luontevasti siirtyä. (Salo 2010: 73.)

Pilvipalveluiden hyödyt on helposti havaittavissa ja sen tarjoama potentiaali on kiistaton. Lisääntynyt joustavuus, helppo palveluiden käyttöönotto ja palvelun ajantasaisuus ja skaalautuvuus ja mahdollisuus muodostaa yhteys palveluun mistä

tahansa, missä on Internet-yhteys, ovat osa pilvipalveluiden konkreettisista hyödyistä. Kyseenalaisempia hyötyjä ovat kustannussäästöt ja palvelun laatu verrattuna oman organisaation aiempaan toimintaan, jotka ovat enemmän tapauskohtaisia ja selviävät, kun yritys vertaa tarkemman omia käytäntöjään uusiin ekstensiivisen analyysin kautta. Uusia sovellusmahdollisuuksia syntyy myös teknologian kehittyessä. Ihmisten käyttäytymisen erilaisten tunniste- ja päätelaitteiden kanssa tarjoaa uusia analyyttisia ja toiminnallisia sovellusmahdollisuuksia. Myös aiemmin haastavat ja aikaa vievät laskennalliset tehtävät nopeutuvat, kun resursseja on tarjolla hetkellisesti suuria määriä. (Salo 2010: 79-83.)

Alan nuoruus aiheuttaa omalta osaltaan tiettyjä huolen aiheita yrityksissä, ja tiettyjä riskejä pilvipalveluiden käyttöön löytyy. Palvelun turvallisuus, saatavuus, ja suorituskyky ovat suurimpia huolenaiheita IDC:n vuonna 2009 yrityspaneelille suuntaamassaan kyselytutkimuksessa. Myös räätälöinti ja integrointi omaan järjestelmään sopivaksi koetaan aiheuttavan haasteita. Ongelmat eivät kuitenkaan ole ylitsepääsemättömiä. Selkeämmät ja tarkemmat SLA:t eli palvelusopimukset mahdollistavat tarkemman riskinhallinnan ja selkeyttävät palveluntarjoajan vastuuta. Alan nuoruuden vuoksi käristään myös paljon ns. lastentaudeista, joita voidaan havaita yleisesti uusien liiketoimintamallien yhteydessä. (Salo 2010: 100.)

Markkinoiden ja palveluiden hakiessa muotoaan yleensä havaitaan standardien vahvistuvan ja erottuvat toimimattomista toimintamalleista. ”Uusien ilmiöiden kohdalla on tapana yliarvioida muutosten vaikutusta lyhyellä aikavälillä ja aliarvioida pitkällä” (Salo 2010: 164). Innovatiivisimpien yritysten ottaessa pilvipalveluita käyttöön käyttäjämäärät kasvavat, pilvipalvelut saavat lisää uusia asiakasreferenssejä, jotka lisäävät luottamusta, ja jotka luovat itseään ruokkivan kehän uusien asiakkaiden hankinnan kannalta. (Salo 2010: 164.)

2 Menetelmät ja työvälineet

2.1 Ohjelmistotuotanto

Ohjelmistotuotanto on yhteisnimitys kaikille niille tuotannollisille prosesseille, joiden tarkoituksena on tietokoneohjelman tai ohjelmistojen tuottaminen. Se koostuu useista eri prosesseista, jotka ovat kytköksissä ohjelmistoprojektin loppu- ja sivutuotoksiin. (Haikala 2004: 16.) Ohjelmistotuotanto on hyvin erilainen tuotannonala verrattuna esimerkiksi perinteiseen teolliseen tuotantoon. Se sisältään monia samoja käsitteitä, mutta verrattaessa esimerkiksi rakennusteollisuuteen voidaan havaita selkeitä eroavaisuuksia näiden kahden alan välillä. Kun tiedostaa ja ymmärtää ohjelmistotuotannon tiettyjä ns. lainalaisuuksia, on helpompi oppia välttämään niihin kytköksissä olevia ongelmia.

Ohjelmistotuotannon ominaispiirteitä

Ohjelmistotuotannolle on ominaista tuotteiden monimutkaisuus. Erityisesti suuret ohjelmistot ratkaisevat usein hyvin monimutkaisia ongelmia, joten niiden toteutus ei myöskään usein ole kovin yksinkertaista. Useista algoritmeista ja niiden välisistä vuorovaikutuksista koostuvat laajalle ulottuvat järjestelmät ovat tietyllä tasolla hyvin abstrakteja, ja kokonaisuutta voi olla ensi silmäykseltä hankala hahmottaa. Monimutkaisuutta pyritään minimoimaan luomalla pienempiä osakokonaisuuksia ja selkeyttämällä niiden välisiä suhteita. Monimutkaisuutta harvemmin voidaan kuitenkaan eliminoida kokonaan, ja usein se on suurin taustalla vaikuttava tekijä. (Haikala 2004: 28.)

Tuotteen tietyntyyppinen aineettomuus ja näkymättömyys vaikeuttavat ohjelmiston projektinhallintaa. Valmiusastetta voi olla hankala arvioida tuotettujen dokumenttien, mukaan lukien ohjelmakoodin, määrästä. Jos esimerkiksi suunnittelu on tehty erityisen hyvin ja selkeästi, voi se tarkoittaa, että yli 50 % projektista on tehty, vaikka riviäkään koodia ei ole vielä tuotettu. (Haikala 2004: 29.)

Ohjelmistojen muutettavuus on verrattain helppoa. Tämä mahdollisuus aiheuttaa muutospaineita ohjelmistolle sen koko elinaikana ja erityisesti kehitysvaiheessa. Jos kesken projektin havaitaan, että asiat voidaankin tehdä paremmin tai tulisi tehdä toisin syystä tai toisesta, muutoksiin tartutaan huomattavasti helpommin yksinkertaisesti siitä syystä, että niihin tarttuminen on niin helppoa. Vanhan toteutuksen voi jättää vierelle

odottamaan ikään kuin varavaihtoehtona, eikä se vie tilaa tai resursseja uudelta. Rakennusteollisuudessa esimerkiksi kerrostalon uudelleenrakennus on täysin erilaisessa asemassa ohjelmiston uudelleenstrukturointiin nähden. (Haikala 2004: 29.)

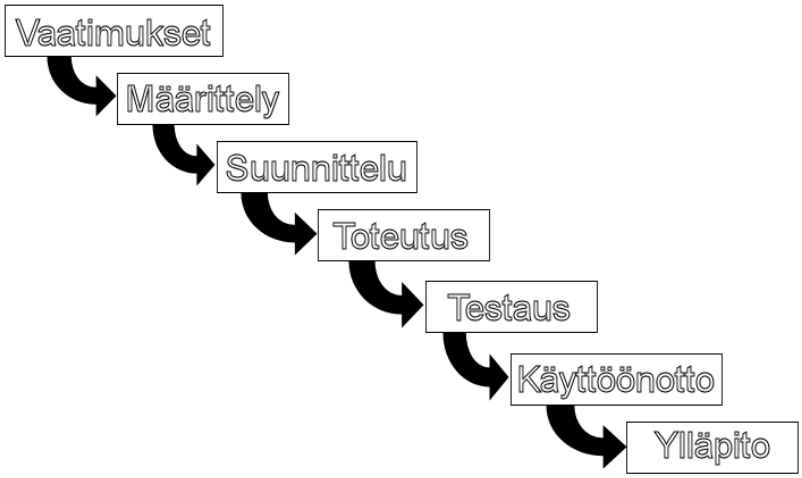
Ohjelmistotuotteita leimaa niiden ainutkertaisuus. Voi olla, että samanlaista toteutusta ei ole koskaan aikaisemmin tehty. Tämä vaikuttaa oleellisesti projektin alussa toteutettavaan suunnitteluun, koska hyvin usein joudutaan aloittamaan ns. tyhjältä pöydältä. Ainutkertaisuus on kytköksissä alan nuoruuden kanssa. Uutta teknologiaa ja tekniikoita tulee kehittäjien tarjolle jatkuvalla syötöllä, joten asiat myös pyritään tekemään parhaalla mahdollisella tavalla. Usein uusin toimivaksi havaittu tekniikka on vaihtoehto, johon monet taipuvat. Tekniikoita ja menetelmiä tulisikin hyödyntää kunkin yksilöllisen projektin vaatimalla tavalla. (Haikala 2004: 30.)

Käytettävien menetelmien skaalautumattomuus on myös ratkaiseva osatekijä ohjelmistotuotannossa. Pienessä projektissa käytetyt menetelmät eivät välttämättä enää toimi järkevästi seuraavassa suuremmassa projektissa. Pitää tunnistaa kullekin projektille siihen parhaiten sopivat menetelmät ja osata soveltaa niitä joustavasti tilanteen mukaan. Ohjelmistojen käyttäytyminen on myös usein hyvin epäjatkuvaa. Monimutkaisen rakenteensa vuoksi, pienimmänkin lenkin hajoaminen saattaa kaataa isonkin järjestelmän. Lenkkejä ollessa hyvin monta, kaikkien niiden toiminnan varmistaminen on vaativa ja monimutkainen prosessi. (Haikala 2004: 30.)

Ohjelmistotuotteiden monimutkaisuus on selkeästi yksi sen ominaispiirteistä voimakkain. On hyvä miettiä ohjelmistoprojektikohtaisesti, mitä ongelmaa itse asiassa ollaan ratkaisemassa. Ongelman syvällinen ymmärtäminen helpottaa monimutkaisen rakenteen strukturoimisessa, ja se on välttämätöntä laadukkaiden tuotteiden ja palveluiden toteuttamiseksi. Tarjolla olevia menetelmiä on useita, joten on oleellista ymmärtää menetelmien tarjoama lisäarvo ja osata hyödyntää niitä oikeissa tilanteissa. Lopulta voidaan havaita ohjelmistoprojektin hyvin elävä luonne. Projekti muuttuu selkeämmäksi sen edetessä, ja näin ollen myös projektin parissa työskentelevien henkilöiden tulee tiedostaa se. Ohjelmistoprojekti eivät ole muuttumattomia, joten oikeanlaisella suhtautumisella projektin suunnittelussa alusta lähtien voidaan jättää pelivaraa muutoksille, kun ne lopulta ilmaantuvat. (Haikala 2004: 31-32.)

Ohjelmistotuotannon vaiheistus

Perinteisen tuotantomallin (kuvio 1) mukaan ohjelmistotuotantoprojektissa käytävät kehitysvaiheet voidaan hahmottaa ns. vesiputousmallin mukaan. Kyseisessä mallissa ohjelmistoprojekti etenee lineaarisesti seuraten tiettyjä vaiheita ja prosesseja, jotka useimmiten ovat mukana ohjelmistoprojekteissa. Nämä prosessit tai kehitysvaiheet pääpiirteittäin ovat projektin alussa tehtävä vaatimusanalyysi sisältäen suunnittelua ja määrittelyä. Suunnitelmien ollessa selkeitä voidaan siirtyä niiden toteuttamiseen. Lopuksi ohjelmistoa testataan ja kun ollaan tyytyväisiä sen tarjoamaan toiminnallisuuden laatuun, tuote julkaistaan. Julkaisun jälkeen seuraa ohjelmistotuotteen ylläpito ja mahdolliset jatkokehitys suunnitelmat. Joskus mallin alkuun mainitaan myös esitutkimus, jonka tarkoitus on korostaa suunnittelun oleellisuutta projektin osana ja sen merkitystä projektin onnistumisen kannalta. (Haikala 2004: 35-37.)



Kuvio 1. Vesiputousmalli

Yllä havainnollistettua ohjelmistokehitysmallia kutsutaan vesiputousmalliksi, ja se kuvastaa ideaalista tuotantomallia, jossa yhden vaiheen päätyttyä siirrytään seuraavaan vaiheeseen, sillä oletuksella, että edelliseen ei tarvitse enää palata. Yllä mainitut prosessin vaiheet ovat osana jokaista ohjelmistoprojektia riippumatta siitä, mitä mallia tai menetelmää siinä käytetään. Malli on kuitenkin utopististinen, koska oletetaan, että kaikki tieto kyseessä olevasta ohjelmistoprojektista on saatu seuraavaan vaiheeseen siirryttäessä ja tieto on muuttumatonta. Kuten aikaisemmin on todettu, ohjelmistoprojekteissa harvoin on näin. Usein edelliseen vaiheeseen joudutaan

palaamaan, koska projektin edetessä käy ilmi, että ohjelmistoon pitää tehdä muutoksia. Määrittelydokumentaatio on myös aluksi saatettu tehdä virheellisten olettamusten tai käsitysten pohjalta. Käytännössä mallissa nuolet ovat siis kustakin prosessista myös takaisinpäin lohiporrasmaisesti aina määrittelyyn asti.

Mallissa hyöty on siinä, että se esittelee selkeästi ja suoraviivaisesti kaikki ohjelmistokehitysprojektissa olevat tarpeelliset vaiheet. Se ei ole syyttä ollut käytössä vuosikymmenien ajan. Se tarjoaa hyvän ja selkeän rakenteen ja etenemissuunnan, läpinäkyvän lopputuotteen hinnoittelun määrittelydokumentaation perusteella, olettaen että muutoksia ei tule tai oteta vastaan, ja selkeän loppuratkaisun. Vesiputousmallin heikkous on sen kankeudessa, byrokraattisuudessa ja eksessiivisen alku määrittelyn ja dokumentoinnin ylipainottamiseen. Se ei anna tilaa muutokselle ja sille tosiasialle, että projekteilla on tapana muotoutua selkeämmiksi vasta sen edetessä. Liiallinen määrittely heti projektin alkumetreillä voi olla lopulta erittäin turhaa ja vain hidastaa projektin etenemistä.

2.2 Pilvipalvelut ja käyttäjäkeskeisyys

Pilvipalvelut tarjoavat käyttö- ja käyttäjäkokemuksia siinä missä perinteiset operatiiviset järjestelmät ja tietokoneohjelmat. Toisin kuin operatiiviset järjestelmät, pilvipalvelut taistelevat käyttäjistään vapaammalla markkinapaikalla ja ovat paljon alttiimpia käyttäjien palveluista saamansa käyttökokemuksen vaikutuksen seurauksille. Pilvipalvelumarkkinat ovat avoimemmat, palveluita on tarjolla paljon, ne palvelevat useita eri käyttötarkoituksia. Käyttäjät ja yritykset voivat liikkua tuotteesta toiseen paljon vapaammin kuin perinteisten järjestelmien ja tietokoneohjelmien kanssa. Järjestelmät, jotka ovat integroitu osaksi yritystä, on käytetty paljon rahaa. Käyttäjät on koulutettu käyttämään kyseistä järjestelmää ja siitä irtautuminen on lähes mahdotonta tai vähintäänkin erittäin kallista. Pilvipalvelut eivät pysty sitomaan käyttäjiään samalla tavalla, mikä on myös yksi palveluiden houkuttelevuuden keihäänkärjistä. Tästä syystä pilvipalveluiden suunnittelussa käyttäjäkokemus on erittäin oleellisessa osassa palvelun menestyksen kannalta. Käyttäjät pyritään sitomaan palveluun ylivoimaisen käyttäjäkokemuksen ansiosta. (Sinkkonen ym. 2009: 17-18.)

Kaksi oleellista termiä nousevat pilvipalvelun suunnittelussa esiin: palvelun käytettävyys ja sen tarjoama käyttäjä kokemus muodostaa käyttökokemus. Käytettävyys on terminä vanhempi ja ehkä selkeämpi toisin kuin uudempi erityisesti pilvipalveluista puhuttaessa

esiin nouseva käyttäjäkokemus. Käytettävyydellä viitataan siihen kuinka tehokas, hyödyllinen ja kyseessä olevaan käyttöympäristöön soveltuva kyseinen tuote tai palvelu on. Käyttäjä lopuksi määrittää onko tuote tai palvelu käyttökelpoinen. Tehokkuutta voidaan mitata selkeästi ja hyödyllisyyttä arvioida vähintäänkin suuntaa antavasti, mikä on oleellista erityisesti yrityskäytössä. Tärkeää on, että tuote tai palvelu sopii ko. tehtävään sen omassa ympäristössään, ja että se sopii myös sitä käyttäville ihmisille. (Sinkkonen ym. 2009: 18-22.)

Käyttäjäkokemus viittaa suoraan siihen, minkälaisia tuntemuksia palvelun tai tuotteen käyttäminen käyttäjässä herättää. Käyttökokemuksessa itse käyttäjä on suuressa roolissa. Käyttäjän tunteet, odotukset, brändiin liittyvät assosiaatiot ja motivaation palvelun käyttämiseen muodostavat yhden elementin. Palvelu itsessään tarjoaa jonkinlaisen sisällön, toiminnallisuuden ja visuaalisen ilmeen muodostaen jonkinasteisen elämyksen, ja toisen elementin. Nämä kaksi elementtiä yhdistetään ympäristöön, jossa käyttökokemus tapahtuu. Se muodostaa lopullisen käyttäjä kokemuksen. (Sinkkonen ym. 2009: 23-24.)

Erityisesti käyttäjän odotukset on syytä nostaa tästä yhtälöstä esille tarkempaan tutkintaan. Kaikilla käyttäjillä ja ihmisillä on tulevaisuudesta unelmia ja haaveita, jotka muodostavat tulevaisuudenodotukset. Palveluntarjoaja luo ensin käyttäjälle odotuksia palvelun käytöstä ja siihen liittyvistä mahdollisuuksista ja saavutettavista asioista ja unelmista. Nämä assosiaatiot ovat voineet muodostua brändin historiallisen arvon vuoksi tai käyttäjän ensi kertaa vieraillessa palveluntarjoajan sivuilla. Siksi oleellista on, että palvelun tuottama lisäarvo tuodaan selkeästi ja suoraviivaisesti esiin. Nykyhetki on menneisyyden muistojen ja tulevaisuuden unelmien kohtauspaikka, johon palveluntarjoajan on päästävä kiinni. (Sinkkonen ym. 2009: 24.)

Käyttäjäkeskeinen suunnittelu ja kehitys

Käyttäjäkeskeisyyden voisi olettaa olevan itseisarvo kaupallisia ja ilmaisia palveluita suunniteltaessa käyttäjille käytettäväksi. Näin ei kuitenkaan usein ole. Tietotekniikka ja muut insinööritieteet eivät tarjoa menetelmiä parhaan mahdollisen käyttörajapinnan suunnitteluun, koska ko. tieteen ala ei sisällä työvälineitä käyttäjien toimintatapojen tutkimiseen. Esimerkiksi kognitiotiede ja psykologia tarjoavat paremmat menetelmät käyttäjän ymmärtämiseen. Siksi sieltä onkin syytä hakea menetelmiä ongelman tehokkaampaan ratkaisemiseen. (Sinkkonen ym. 2009: 28-29.)

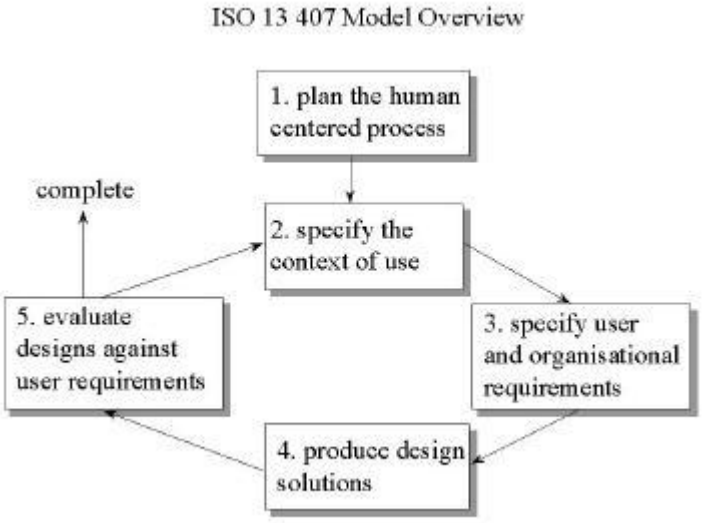
Käyttäjäkeskeinen suunnittelu lähtee siitä, että tunnistetaan palvelun potentiaaliset ja tulevat käyttäjät ja heidän ominaisuutensa: miten he käyttävät palvelua, missä he sitä käyttävät, missä ympäristössä sitä käytetään ja mitä lisäarvoa se heille tuottaa. Nämä ovat muutamia oleellisia kysymyksiä tehokkaan käyttörajapinnan suunnittelun kannalta. Eri käyttötilanteiden hahmottaminen on suunnittelijoidenkin etu, koska se antaa heille syvällisempää ymmärrystä siitä miten noviisi käyttäjä kokee palvelun verrattuna esim. tietojenkäsittelytieteen diplomi-insinööriin tai muuhun teknisen alan ammattilaiseen. Käyttötilanteet voivat vaihdella radikaalistikin, koska esimerkiksi käyttäjän tunnetila, niin kuin ihmisten ylipäätään, voi muuttua nopeasti ja voimakkaasti. Palvelun toteuttavan yrityksen näkökulmasta taustalla on syytä pitää mielessä yrityksen liiketoiminnalliset tavoitteet. (Sinkkonen ym. 2009: 27.)

On useita eri syitä suunnitella ja kehitellä palvelu käyttäjäkeskeisesti. Yrityksen kannalta oleellisin on tietenkin palvelusta yritykseen virtaava raha. Jos käyttäjät pitävät palvelusta ja näkevät sen hyödyllisenä, he käyttävät sitä ja suostuvat maksamaan siitä. Rahan saanti voi merkitä myös säästöjä ja kustannustehokkuutta. Esimerkiksi jos tiedetään kenelle tehdään ja miten palvelu tulisi tehdä, ei tarvitse käyttää yhtä paljon aikaa asiasta keskusteluun palavereissa. Käyttäjät tarvitsevat vähemmän teknistä tukea kun tuote on tehty tarpeeksi ymmärrettäväksi ja helppokäyttöiseksi. Tehokkuus ilmenee, kun palvelu on suunniteltu huolella oikealle kohderyhmälle, jolloin sitä ei tarvitse päivittää jatkuvasti. Myös käyttäjän työ on tehokkaampaa, kun palvelut on suunniteltu oikeasta näkökulmasta. (Sinkkonen ym. 2009: 28-30.)

Kehitysprosessi voidaan jakaa ohjelmistotuotannon periaatteiden mukaan yksinkertaistetusti konseptisuunnitteluun, toteutukseen, testaukseen ja ylläpitoon. Kehitysprosessi on riippumaton käytettävästä ohjelmistotuotannon menetelmästä. Konseptisuunnittelulla tarkoitetaan tässä käyttöliittymäsuunnittelua, ja painopiste suunnittelussa siirretään käyttäjän näkökantaan. Itse toiminnallisuus ja sovelluslogiikka eivät käyttäjäkeskeisessä suunnittelussa ole pää-osassa, mutta luonnollisesti tehokas ohjelma vaatii myös hyvin toteutetun sovelluslogiikan. Aloittamalla kehitysprosessin konseptisuunnittelusta saadaan erittäin hyvä yleiskuva siitä, mitä toiminnallisuutta palvelun pitää sisällyttää. Tarvittavan sovelluslogiikan monimuotoisuuden hahmottaminen on helpompaa, ja myös kustannusarvioiden tekeminen on näin ollen selkeämpää ja perustuu faktoihin. (Sinkkonen ym. 2009: 31-32.)

Suuremmissa kehitysprojekteissa suunnittelu on oleellisessa osassa varsinkin kehitystiimin ollessa suuri ja asianomaisten määrän kasvaessa. Sen vuoksi on oleellista, että kaikki kehitys prosessiin liittyvät henkilöt ovat jatkuvasti tietoisia ja tiedotettuja siitä, mitä tehdään ja miksi. Tämä parantaa oleellisesti tuotannon tehokkuutta, ja vältytään osapuolien omatoimisilta olettamuksilta projektin etenemisen suunnasta tai muodosta. (Sinkkonen ym. 2009: 32.)

Käyttäjäkeskeisen suunnittelun perusidea pohjautuu ISO 13407 –standardiin (Introduction to ISO 13407 1999), joka määrittää perusrakenteen useimmille käytössä oleville käyttäjäkeskeisille suunnittelumenetelmille (kuvio 2).



Kuvio 2. ISO 13407 -standardin mukainen kaavio käyttäjäkeskeisestä tuotekehityksestä (Introduction to ISO 13407 1999).

Mallin mukaan ensin tarkennetaan käyttäjien tarpeet, ja hahmotetaan sitä vastaava sisältö. Sitten määritellään käyttäjien ja organisaation vaatimukset tuotteelle, ja tuotetaan suunnitteluratkaisu. Lopuksi arvioidaan suunnitelmaa käyttäjien vaatimuksiin nähden, ja hyväksytään tai hylätään tuotos. Jos tuotos ei ole miellyttävä ja vaatimukset täyttävä, lopputulosta on syytä parantaa. Välttämättä ei ole aina syytä palata kuvan kohtaan 2, vaan voidaan mennä suoraan kohtaan 4, jos käyttäjät ja heidän vaatimuksensa on uskottavasti tunnistettu. (Sinkkonen ym. 2009: 34)

Käyttäjäkeskeinen sovelluskehitys on ihmisläheinen lähestymistapa tekniseen suunnitteluun ja toteutukseen. Eri menetelmät voidaan nähdä tiivistyvän joukoksi

käytännönläheisiä periaatteita, joita kehitysprojektin olisi syytä noudattaa, jotta saavutettu lopputulos tosiaan palvelee käyttäjää parhaalla mahdollisella tavalla. Sinkkonen esittelee seuraavanlaisia suunnitteluperiaatteita (Sinkkonen ym. 2009: 35).

- ”Palvelun tulisi tukea käyttäjien luonnollisia tapoja tehdä tehtäviään.”
- ”Palvelun navigoinnin tulisi olla käyttäjille selkeä ja tehokas.”
- ”Palvelun pitää olla helppokäyttöinen.”
- ”Palvelun suunnitteluratkaisujen pitää olla yhteneväiset ja johdonmukaiset.”
- ”Palvelussa pitää olla juuri ne toiminnot joita käyttäjä tarvitsee – ei enempää eikä vähempää.”
- ”Palvelun termien tulee ollakäyttäjän käsitemaailmasta tai ne pitää selittää.”
- ”Palvelun suunnitteluratkaisujen pitää olla yhteneväiset ja johdonmukaiset.”
- ”Palvelun sisällön tulee olla relevanttia sen käyttäjälle.”

Yllä mainittuja periaatteita noudattamalla pääsee jo hyvin pitkälle. Ne tukevat ja ohjaavat toteutusta oikeaan suuntaan käytännönläheisesti. Kehitysprojektin edetessä, esimerkiksi yhden iteraation päätyttyä, olisi hyvä katsoa toteutettua osaa ja analysoidaan, pätevätkö ehdot vielä. Näin varmistetaan lopputuotteen syntyminen juuri sellaiseksi kuin oli tarkoituskin.

Koska ohjelmistotuotantoa harjoittavat yritykset yleensä käyttävät jotain menetelmää tai mallia projektiensa tukena ja niiden läpiviemisessä, käyttäjäkeskeinen suunnittelu tulisi saada integroitua siihen mahdollisimman tehokkaasti. (Sinkkonen ym. 2009: 41.) Sinkkonen ym. vertailee kirjassaan Helppokäyttöisen verkkopalvelun suunnittelu vesiputousmallia, RUP-mallia ja ketteriä menetelmiä osana käyttäjälähtöistä suunnittelua, mistä otamme ketterät menetelmät lähempään tarkasteluun.

Tiettyjen ominaispiirteiden vuoksi ketterät menetelmät sopivat omalta osaltaan käyttäjälähtöisen suunnittelun kanssa hyvin yhteen. Nämä ominaispiirteet ovat mm. iterointi, inkrementointi ja asiakkaan sitouttaminen projektiin. Jatkuvan iteraation ja uudistamisen kautta on helpompi saavuttaa haluttu lopputulos, koska asiakkaan on helpompi nähdä, miltä tuotos näyttää ensimmäisen iteraation jälkeen ja tehdä

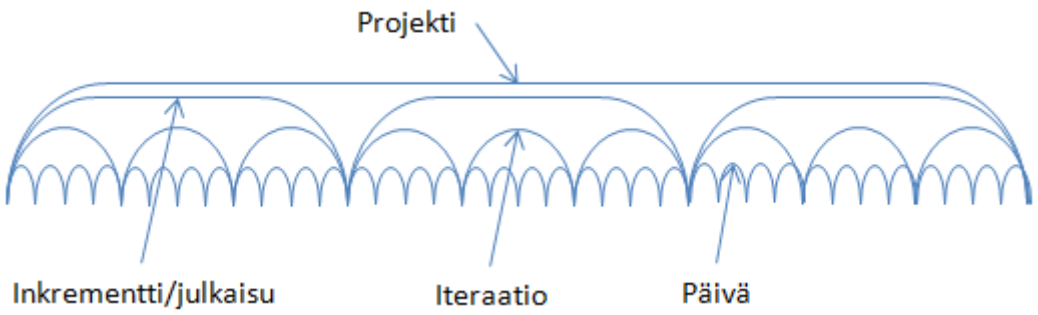
halutessaan muutoksia siihen. Kokonaisuutta rakennetaan inkrementti kerrallaan. Yhdeksi osaksi voidaan määritellä esim. yksi käyttötapaus, jotka asiakas hahmottelee mahdollisesti yhdessä kehitystiimin kanssa selkeäksi kokonaisuudeksi. Asiakas on sitoutettu samalla projektiin, ja näin parannetaan lopputuotoksen laatua, koska asiakas on mukana tekemässä palvelusta sellaista, jonka hän itse haluaa. Palveluun saadaan sisällytettyä kaikista oleellisimmat toiminnallisuudet, ja säästytään sillisalaatilta missä on vähän kaikkea. Lopputuotteen arvostus on oleellista myös asiakkaalle lopputuotteen tilaavana osapuolena. (Sinkkonen ym. 2009: 43-44.)

Ketterät menetelmät asettavat myös tiettyjä haasteita projektille. Koska menetelmät eivät itsessään käsitä käyttäjätutkimusta, asiakkaan on syytä tehdä perusteellinen käyttäjätutkimus ennen projektin aloittamista. Muuten saatetaan päätyä tilanteeseen, jossa tehdään turhia iteraatioita asiakkaan kustannuksella. Perusteellisella tutkimuksella tilanne voidaan välttää, mikä lopulta säästää aikaa ja rahaa. Pilvipalveluiden suunnittelussa kokonaisuuden hahmottaminen on myös erittäin oleellista. Käyttöliittymää, joka toimii päällimmäisenä rajapintana käyttäjälle, ei voi tehokkaasti toteuttaa lisäilemällä siihen sokeasti osia. Käyttöliittymää pitää katsoa kokonaisuutena ja elementtien välisiä suhteita pitää hahmotella suunnitteluvaiheessa ennen niiden lopullista yhteensovittamista. (Sinkkonen ym. 2009: 44-45.)

2.3 Ketterä kehitys

Ketterän ohjelmistokehityksen konsepti on muotoutunut vuosien mittaan ohjelmistotuotannon arkipäiväistyessä, ja se on syntynyt vastineeksi perinteiselle ohjelmistotuotannon käsitteelle. Eri ketterät menetelmät pyrkivätkin määrittelevät eri tavalla sen millaisesta näkökulmasta ja millä ajattelutavalla ohjelmistoprojekti tulisi toteuttaa. Jotkut menetelmät ovat enemmän projektin hallintaan suuntautuneita, ja toiset taas hyvin ohjelmointikeskeisiä. Eri menetelmiä pyritään yhdistelemään parhaan mahdollisen lopputuloksen saavuttamiseksi.

Ketteriä menetelmiä kuvataan usein iteratiivisiksi ja inkrementaalisiksi. ”Inkrementillä tarkoitetaan tarkoitetaan toiminnallisuuden palasta, mikä on tarvittaessa käyttöönotettavissa. Iteraatiolla taasen tarkoitetaan yhtä sykliä, jonka aikana inkrementti valmistetaan” (Ketterät käytännöt — Iteraatiot ja inkrementit 2011). Kuviossa 3 on kuvaus iteraatioiden ja inkrementtien osuudesta kehitysprojektissa.



Kuvio 3. Kehityssyklit, iteraatiot ja inkrementti. (Ketterät käytännöt — Iteraatiot ja inkrementit 2011)

Kaikkien ketterien kehitysmallien perustava ajatus nojaa ketterään manifestiin (eng. Agile Manifest), jonka teki ryhmä ohjelmistoalan vaikuttajia. Kyseinen dokumentti on hyvin tunnettu julkilausuma ohjelmistotyön parissa työskenteleville. (Principles behind the Agile Manifesto 2001.)

Agile Manifest kiteyttää ketterän ohjelmistokehityksen ydinsanoman neljäksi arvovertaukseksi, jossa pyritään arvostamaan enemmän

- yksilöitä ja kommunikointia kuin prosessien ja työkalujen
- toimivaa ohjelmistoa kuin kattavan dokumentaation
- asiakas yhteistyötä kuin sopimusneuvottelujen
- muutoksiin mukautumista kuin suunnitelman noudattamista.

Manifesti korostaa, että vaikka oikeanpuoleisillekin käsitteillä annetaan arvoa, on se pienempi verrattuna vasemmalla oleviin. Arvot sanelevat, mitä ohjelmistotuotannossa kokonaisvaltaisena prosessina pitäisi arvostaa. Arvoista johdetaan periaatteet, miten konkreettisesti toimia, ja ne voidaan jalostaa vielä yksityiskohtaisimmiksi käytännöiksi. Ketterät menetelmät eivät siis tarkoita sitä, että rationaalinen ajattelutapa ja projektinhallinta tulisi unohtaa vaikka käytettäisiinkin modernimpaa ja joustavampaa kehitysmallia. (Principles behind the Agile Manifesto 2001.)

Ketterän kehityksen ajatuksena on, että menetelmän on tulisi olla projektikohtainen ja eri menetelmillä voi päästä haluttuun lopputulokseen. Agile Manifestiin ei halua ottaa kantaa menetelmien käytäntöihin, vaan siihen on kirjattu 12 yleispätevää ketterän ohjelmistokehityksen periaatteita, jotka kuuluvat jokaiseen kehitysmalliin. Näitä

noudattavat kaikki ketterät menetelmät, ja ne ovat ohjenuora sille, miten ketterän ohjelmistoprojektin tulisi edetä. Eri kehitysmalleilla voi olla myös yksilöllisiä periaatteita, jotka pohjautuvat niiden omiin arvomääritelmiin, ja ovat yksilöllisiä riippuen menetelmästä. Eri menetelmät muodostuvat siis itse manifestin arvoista ja niistä johdetuista periaatteista ja niiden yksilöllisistä arvoista, periaatteista ja käytännöistä, jotka ovat muodostuneet edellä mainittujen pohjalta. (Principles behind the Agile Manifesto 2001.)

Ketterä kehitys pitää sisällään kaikki perinteisen vesiputousmallin sisältämät ohjelmistotuotannon vaiheiden osa-alueet, mutta niihin pyritään suhtautumaan yhtenä kokonaisuutena eri näkökulmasta. Yleisesti ketterällä kehityksellä pyritään viittaamaan siihen, että toisin kuin perinteisen vesiputousmallin mukaan, näitä vaiheita tulisi voida edetä ketterästi, lukkiutumatta ennalta valittuun tiettyyn ajattelu- ja toteutustapaan vain sen itsensä vuoksi. Ohjelmistotuotannossa eri osa-alueet ovat niin tiiviisti nidottuina toisiinsa, että niitä on hankala erotella tehokkaasti ja edetä suoraviivaisesti vaihe vaiheelta. Kaikki osa-alueet ovat läsnä projektin joka vaiheessa, ainoastaan fokus tiettyjen osa-alueiden välillä muuttuu projektin edetessä. Ohjelmistoprojektit ovat monimutkaisia ja moniulotteisia, joten myös käytettävät menetelmät tulisi sovittaa kyseessä olevaan projektiin, eikä päinvastoin.

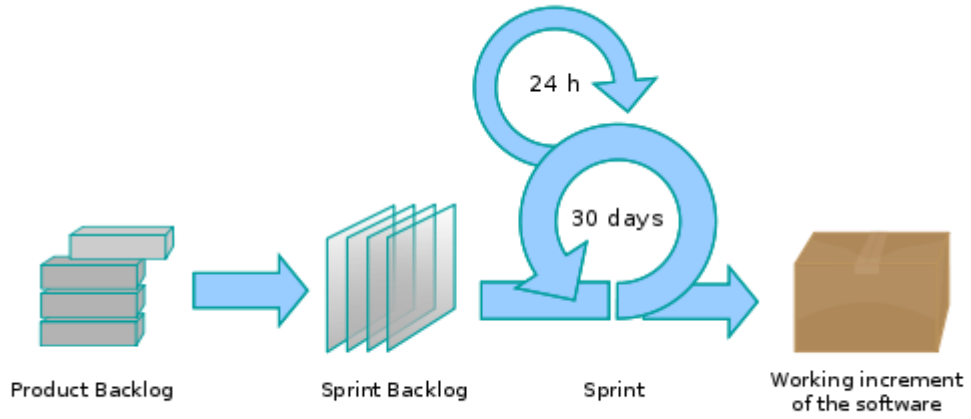
2.3.1 Scrum

Scrum on projektiorientoitunut kehitysmenetelmä. Sillä pyritään hallinnoimaan ohjelmistoprojektin etenemistä mahdollisimman tehokkaasti, eikä se ota kantaa matalan insinööritason käytäntöihin. Scrum-projektissa on kolme pääroolia ja ne ovat asiakas eli tuotteen omistaja, Scrum-mestari ja kehitystiimi. Tuotteen omistaja määrittelee mitä ominaisuuksia ja toiminnallisuutta tuotteen tulisi sisältää. Scrum-mestari on periaatteessa projektipäällikkö, joka valvoo projektin kehitystä sekä mahdollistaa kehitystiimin optimaalisen toiminnan ratkomalla projektia hidastavia ongelmia. Kehitystiimi vastaa itse ohjelman toteuttamisesta. Se toimii itsenäisesti, ja tiimi kootaan tarvittavien osaamisprofiilien perusteella, että ohjelma voidaan toteuttaa. Tällä korostetaan yhteisvastuullisuutta ja kaikkien tasa-arvoisuutta. Tehtävät jakautuvat tiimin sisällä luonnollisesti kunkin osaamisen perusteella. (Ketterät käytännöt — Scrum 2011.)

Scrum poikkeaa tavanomaisen määriteltyjen käytäntöjen mallin perinteestä ja antaa käyttäjälleen arvoja, joihin aktiviteetit ja periaatteet nojaavat. Scrumin arvot ovat sitoutuminen, keskittyminen, avoimuus, kunnioitus ja rohkeus. Sitoutuminen viittaa kehitystiimin vastuuseen osana ohjelmistoprojektia. Perinteisimmissä malleissa kehitystiimille annetaan tehtävät ja menetelmät valmiiksi määriteltynä pakettina ja heidän osansa on toimia mekaanisena välikappaleena toiminnallisuuden toteuttamisessa. Tässä kehitystiimin vastuulla on toiminnallisuuden toteuttaminen itse parhaaksi näkemällään tavalla ennalta määrätyssä ajassa. Tehtävä muotoillaan sellaiseen muotoon, että lopputuloksesta voidaan selkeästi arvioida onko tiimi onnistunut vai ei. (Ketterät käytännöt — Scrum 2011.)

Keskittyminen kuvastaa sitä, että ns. sprintille eli yhdelle iteraatiolle asetetaan aina selkeä tavoite, joka on saavutettavissa, sprintin aikana. Silloin ei ole syytä keskittyä mihinkään muuhun kuin itse tavoitteen saavuttamiseen, kaikki muut esiin ilmaantuvat asiat jäävät myöhemmäksi. Projektityö pyritään pitämään mahdollisimman avoimena myös ulkopuolisille. Kaikki dokumentit ovat kaikkien halukkaiden saatavilla, ja myös ulkopuolisilla on mahdollista osallistua kokoukseen, joskin vain tiimin jäsenillä on oikeus käyttää puheenvuoroja. Tämä edistää projektin läpinäkyvyyttä ja ihmisten välistä kommunikointia. (Ketterät käytännöt — Scrum 2011.)

Ihmisten kunnioitus on oleellista toimivassa tiimissä, jossa usein samat ihmiset työskentelevät toistensa parissa. Pyritään luottaman siihen, että kaikki tekevät parhaansa kykyjensä mukaan. Rohkeus tehdä ja olla kaikkia edellisistä arvoista ei voi aliarvioida. Paras lopputulos saavutetaan, kun uskalletaan toimia projektin parhaaksi ja osataan avoimesti myöntää omat henkilökohtaiset virheet, jotta kaikki voivat niistä vastavuoroisesti oppia. (Ketterät käytännöt — Scrum 2011.) Kuviossa 4 havainnollistetaan tarkemmin Scrum-projektin etenemistä.



Kuvio 4. Scrum-projektin eteneminen (Scrum process 2009).

Scrum-projektin eteneminen voidaan havainnollistaa siinä tapahtuvien aktiviteettien kautta. Projekti aloitetaan visioinnilla eli hahmotellaan, minkälaista tuotetta ollaan tekemässä ja mitä projektilta halutaan saavuttaa. Tätä voisi verrata vesiputousmallin ensimmäisiin alustaviin vaiheisiin ja esitutkimukseen. Tämän jälkeen muodostetaan alustava työlista, jossa määritellään ominaisuuksia ja toiminnallisuutta, joita tuotteen tulisi pitää sisällään. Kun Scrum-mestari ja tiimi tuotteen omistajan kanssa ovat luoneet ominaisuuslistan, tuotteen omistaja priorisoi listalla olevat tuotteet ennen varsinaisen toteutuksen alkamista. Tämä on yllä olevan kuvion ensimmäinen vaihe, product backlog. (Ketterät käytännöt — Scrum 2011.)

Ominaisuuslistan pohjalta kehitystiimi suunnittelee tulevan sprintin ja arvioi kuinka monta ominaisuutta se listalta saa toteutettua sprintin aikana (yllä olevan kuvion sprint backlog). Ominaisuudet valitaan pääsääntöisesti niiden prioriteetin mukaan, jotta kaikkein tärkeimmät asiat saadaan toteutettua ensimmäisinä. Sprintille määritetään myös selkeä tavoite, joka mahdollistaa sprintin arvioinnin sen päätyttyä. Seuraavaksi toteutetaan itse sprint. Tänä aikana kehitystiimillä on vapaat kädet toteuttaa valitut ominaisuudet ja organisoida oma toimintansa tavoitteen saavuttamiseksi. Tänä aikana mitään muutoksia tai ehdotuksia ei oteta vastaan vaan tiimi keskittyy sprintin suunnittelussa määriteltyihin tehtäviin. (Ketterät käytännöt — Scrum 2011.)

Sprintin ollessa käynnissä pidetään päivittäin tilannekatsaus eli daily scrum, jossa kehitystiimin jäsenet kertovat, mitä tekivät edellisenä päivän, mitä suunnittelivat tekevänsä tänään, ja arvioivat onko asioita, jotka hidastavat heidän toimintaansa.

Tilaisuus on kaikille avoin, mutta vain kehitystiimin jäsenet puhuvat ja kertovat toiminnastaan, ja siellä kehitystiimin jäsenet vastaavat vain edellä mainituista asioista. Jos muita keskustelunaiheita ilmenee, niistä pidetään palaveri erikseen niiden henkilöiden kesken, joita asia koskettaa. Sprintin päätyttyä pidetään katselmus, jossa tutkaillaan tiimin aikaansaannoksia ja esitetään ne tuotteen omistajalle. Suuremmissa kokonaisuuksissa sprinttejä tehdään useampia, koska ominaisuuksia on enemmän. Tuotteen omistaja lopulta hyväksyy tai ns. hylkää tuotteen, mistä seuraa uusi sprint, ja tuotteen kehitys jatkuu. Päämääränä sprinteillä on lopulta käyttöönotettava tuote. (Ketterät käytännöt — Scrum 2011.)

Katselmuksen jälkeen suoritetaan jälkitarkastelu, jossa analysoidaan itse kehitysprosessia. Scrum-mestari, tiimi ja tuotteen omistaja kertovat mikä prosessissa meni hyvin ja mikä huonosti. Tämän jälkeen alkaa uuden sprintin suunnittelu ja prosessi alkaa alusta. Työlista on taas vapaasti tuotteen omistajan muokattavissa, ja lopuksi tiimi sitoutuu toimittamaan hyväksymänsä ominaisuudet. (Ketterät käytännöt — Scrum 2011.)

2.3.2 Ketterä sovelluskehitysprojekti käytännössä

Sovellukselle asetettavat vaatimukset ja määrittely

Sovelluskehitys projekti alkaa hyvin usein käytettävästä tuotantomallista riippumatta suunnittelulla. Kehitysprojektin aluksi on tarkoitus saada yleiskuva siitä, minkälaista sovellusta ollaan tekemässä ja minkälaista tarkoitusta sen on määrä palvella. Suunnittelussa on otetta huomioon sovellukselle asetettavia vaatimuksia liittyen mm. sen ylläpitoon ja jatkokehitykseen. Yleisesti sovelluskehitysprojekteissa sovellusvaatimukset ja määrittelyt käydään asiakkaan kanssa läpi, jos sovellus on tarkoitus tehdä räätälöitynä tuotteena integroitavaksi asiakkaan omiin tietojärjestelmiin. Samanlaista suunnitteluperiaatetta voidaan käyttää myös yritysten sisäisissä kehitysprojekteissa, ns. massatuotantoon tarkoitetuissa sovelluksissa ja pilvipalveluiksi tarkoitetuissa sovelluskehitysprojekteissa. (Kuha 2008: 30.)

Projektin alussa tehtävien määrittelyiden on tarkoitus kuvastaa karkealla tasolla sitä, mitä toiminnallisuutta sovelluksen on tarkoitus tarjota. Projektin alussa tapahtuvassa määrittelyissä ei yleensä ole syytä mennä liian yksityiskohtaisiin vaatimuksiin, koska sovelluskehitysprojektit elävät hyvin voimakkaasti. Usein kehitysprojekteissa määrittelyvaiheessa ei pystytä vielä hahmottamaan koko sovellukseen sisällytettäväksi

haluttua toiminnallisuutta, ja uudet ideat ja parannusehdotukset selkenevät vasta projektin edetessä. Sovelluskehitys on empiirinen prosessi, ja alustavasti tehdyt löyhemmät vaatimukset mahdollistavat ketterämmän reagoinnin uusiin ideoihin. (Kuha 2008: 32.)

Kommunikointi asiakkaan ja sovelluskehityksen toteuttavan organisaation välillä ja yrityksen sisällä sovelluskehitystiimin keskinäinen kommunikointi on oleellisessa osassa erityisesti projektin alkuvaiheilla. Kehitystiimin tulee osata kommunikoida tehokkaasti asiakkaan kanssa ja ymmärtää asiakas yrityksen toiminnan luonne: mitä erilaisia käsitteitä ja kokonaisuuksia yrityksen toimintaa liittyy. Alustavat vaatimukset ovat ensimmäinen iteraatio eli sprint, joka luo pohjaa tulevalle kehitykselle. Sen aikana on hyödyllistä saada joitakin perusasioita liittyen kehitysprojektiin. Näitä voisivat olla mm.

- alustava toimintolista
- sovellusalamalli (tapauskohtaisesti)
- arkkitehtuurimalli (tapauskohtaisesti)
- malleja/Luonnoksia oleellisimmista käyttötapahtumista.

Alustavan toimintolista kuvastaa konkreettisia toimintoja, joita sovelluksella voidaan tehdä. Esimerkkeinä varastointihallintajärjestelmässä voisi olla tuotteiden kirjaus varastoon, tuotteiden keräys varastosta ja tuotteiden lähetys. Perinteisesti ohjelmistotuotantoon ja sen suunnitteluosuuteen kuuluu erilaisia kaavioita, joilla pyritään kuvaamaan sovellusta graafisessa, helpommin havainnollistettavassa muodossa. Nämä sekvenssi-, luokka- ja käyttötapauskaaviot yms. voivat olla oleellinen osa kehitystyötä, jos ne tuottavat lisäarvoa sovelluksen kehityksen näkökulmasta. Usein tämä tieto, jota kaavioilla pyritään kuvaamaan, löytyy muistakin lähteistä. Tällöin kaaviot eivät tuota mitään lisäarvoa projektin näkökulmasta. Kaavioiden perimmäinen tarkoitus on edistää tiedonkulkua ja kommunikointia sovelluksen kehityksessä olevien henkilöiden kesken. (Kuha 2008: 32-33.)

Sovellusprojektin aloitus

Sovellusprojektin aloitus ja alustus ovat mekaaninen prosessi, joka suoritetaan uutta kehitysprojektia aloitettaessa. Tarkoitus on tehdä päätöksiä mm. siitä, minkälainen

kehitysympäristö luodaan, ja arvioida minkälaista mallia projektissa tulisi käyttää. Kehitysympäristöstä ja valituista kehitystyökaluista riippuen luodaan ensin pääprojekti, jonka alle luodaan tarvittaessa uusia aliprojekteja. Aliprojektit ovat omia kokonaisuuksiaan ja linkittyvät pääprojektin alle muodostaen lopulta toimivan sovelluksen. Aliprojekti voi olla esimerkiksi toiminnallisuuden jokin suurempi kokonaisuus, kuten sovelluksen tietorakenne, joka voidaan hyvin ja luonnollisesti erottaa erilliseksi aliprojektikseen. Välttämättä projektin aluksi ei vielä tiedetä aliprojektin tarkkaa lukumäärää, ja niitä perustetaan tarpeen mukaan. (Kuha 2008: 72-73.)

Muutamat oleelliset asiat ovat aina mukana jokaisessa sovelluskehitysprojektissa. Näihin komponentteihin lukeutuvat mm. asianmukainen sovelluskomponenttien testausympäristö ja lokien kirjoittamiseen soveltuva apukirjasto. Ilman toimivaa ja tehokasta sovelluskomponenttien testausta ja lokitiedostoja sovelluskehitys on hyvin aikaa vievää ja tehotonta. (Kuha 2008: 77.)

Sovelluksen testaus ja lokitiedostojen kirjoittaminen kulkevat käsi kädessä sovelluskehitysprojektin alusta loppuun. Eri komponentteja testatessa, virheilmoitukset kirjoitetaan lokiin, josta lopputuloksia voidaan tulkita ja analysoida. Testaus ja lokiin kirjoittaminen voidaan toteuttaa monella eri tavalla, mutta yleisesti esimerkiksi lokiin kirjoittamisen halutaan tapahtuvan hyvin hallitusti ja suorituskykyisesti, jotta se ei vie liikaa resursseja ohjelmaan nähden. Tunnetuimpia lokikirjastoja ovat Log4j ja Apache commons-loggin, jotka ovat yleisesti ja vapaasti saatavilla ja suosituimpia lokikirjastoja Javan kehitysympäristössä. Testausta voi myös suorittaa hyvin sovelluskehityksen lomassa käyttäen puhtaasti valmistuvaa sovelluskoodia testauksen pohjana. Testausta onnistuu myös testaukseen tarkoitettua sovelluskehystä hyödyksi käyttäen, mikä on erittäin suositeltavaa suurempia sovelluskokonaisuuksia rakennettaessa. (Kuha 2008: 77-81.)

Sovelluksen liiketoiminta ja tietokantalogiikka

Sovellusta tehtäessä olio-ohjelmointikielellä, koodia pyritään kirjoittamaan hyvin objekti orientoituneesti ja helposti hahmotettavissa olevien kokonaisuuksien kautta. Objekti voi olla lähestulkoon mikä tahansa käsite, yleismaallinen tai liiketoiminnallinen, jonka voi purkamaan kokoelmaksi ominaisuuksia ja toiminnallisuutta. Liiketoimintalogiikka voi

jakautua useampaan liiketoiminnalliseen käsitteeseen, ja tietokantalogiikka on yleensä oma kokonaisuutensa. (Kuha 2008: 218.)

Määrittelyvaiheessa on selkeytetty eri liiketoiminnalliset käsitteet, ja logiikkaa rakennettaessa niistä tehdään luokkia. Objekti on tämä käsite ja sitä vastaan on luokka, joka sisältää objektin ominaisuuden attribuutteina ja toiminnallisuuden funktioina. Logiikkaa rakennettaessa toteutuksen lisäksi on pohdittava objektien keskinäiset yhteydet ja tiettyjä teknisiä yksityiskohtia, jotka vaikuttavat ohjelman ylläpidettävyyteen ja muunneltavuuteen. Kuten reaalimaailmassakin objektit ovat harvoin omia yksittäisiä, toisista riippumattomia entiteettejään. Objektien olemassaolo perustuu siihen, miten ne ovat liitoksissa muihin. Tämä ilmenee olio-ohjelmoinnissa perintänä. On olemassa yläluokkia ja niitä vastaavia alaluokkia. Alaluokat tyypillisesti perivät yläluokkien ominaisuudet ja toiminnallisuuden sisältäen joukon omia ominaisuuksiaan tai toiminnallisuuttaan. Näin yläluokan käsitteestä saadaan jalostettua useita alaluokan käsitteitä, jotka ovat selkeästi liitoksissa yläluokkaansa.

Pienemmissä ja kohtalaisen yksinkertaisissa ohjelmistokokonaisuuksissa, joissa työskentelee vain muutamia ihmisiä kehitystiimissä, pelkkä luokkien määrittely ja toteutus voi hyvin riittää. Suuremmissa ohjelmistokokonaisuuksissa on myös syytä määritellä yleinen rajapinta, mitä toiminnallisuutta sitä implementoivat luokat sisältävät. Tämä mahdollistaa sen, että kaikki kehitystiimin jäsenet tietävät miten ohjelman tulisi päätasolla kommunikoida ulospäin, tarvitsematta tietää toisten kehitystiimin jäsenten toimista. Näin ei synny päällekkäisyyttä tai ristiriitaisuutta kooditasolla. Ohjelmistorajapinnat eli Application Platform Interface on yksi erimerkki rajapinnasta, jossa tietyn kaupallisen tai avoimen ohjelmistokirjaston funktioita käytetään hyödyksi sen tarjoaman rajapinnan kautta. (Kuha 2008: 219.)

Tietokantaoperaatiot ovat oleellisessa osassa useissa ohjelmistokehitysprojekteissa. Niiden toteutus ei juuri poikkea liiketoimintalogiikan toteutuksesta. Suunnittelu- ja toteutusmalleja tietokantaoperaatioiden kehittämisessä on myös useita, yksi yleisimmin käytetyistä on DAO-malli. DAO eli Data Access Object viittaa ydinsovelluslogiikan ja tietokannan väliseen tiedonvaihtoon ja sen manipulointiin. Perinteisessä mallissa, jossa ei varsinaista mallia edes ole, on tapana ottaa yhteys tietokantaan suoraan ohjelmiston liiketoimintalogiikasta. Tietokantaoperaatiot upotetaan suoraan ohjelmakoodiin tarvittaviin paikkoihin ja toteutus sekä ulos saatava tieto räätälöidään tilanteen

vaatimalla tavalla. DAO-mallissa tietokantaoperaatiot eriytetään omaksi osakseen yrityksen ydinlogiikasta. Tietokantaoperaatiot kerätään yhteen yhden tai useamman luokan alle ja niiden toimintaa hallinnoidaan yhdestä paikasta. Tämä tuo selkeitä etuja sovelluksen kehityksen kannalta ja tiedonhaku saadaan eriytettyä itse yrityslogiikasta. (Kuha 2008: 186.)

Sovelluslogiikka jakautuu sovelluksen sisäisiin palveluihin ja ulkoisiin palveluihin. Sisäiset palvelut ovat toimintoja, joita ulkoiset palvelut kutsuvat suorittaessaan omaa toiminnallisuuttaan. Jos moni ulkoinen palvelu pyrkii suorittamaan samanlaista toiminnallisuutta rutiininomaisesti, siitä on syytä tehdä itsenäinen sisäinen palvelu. Näin vältytään toistolta, joka muuten esiintyisi usean palvelun eli funktion kutsuessa lähes samanlaista toiminnallisuutta erikseen omassa koodissaan.

Sovelluksen ulkoasun rakentaminen

Sovelluksen ulkoasun eli itse käyttöliittymän rakentaminen on tärkeä osa kaikkia sovelluksia. Käyttöliittymä on asiakkaan liitäntärajapinta sovelluksen liiketoimintalogiikkaa, ja käyttäjät usein arvioivatkin sovellustuotteita sen ulkoasun perusteella, koska se on ainoa ohjelman konkreettinen elementti, jonka voi nähdä ja jonka kanssa toimitaan.

Käyttöliittymä voidaan jakaa eri elementteihin. Ulkoasun rakenne, yksittäiset käyttöliittymän elementit, visuaalinen ulkoasu ja sisältö ovat omia selkeitä kokonaisuuksiaan, joista ulkoasu kokonaisuudessaan rakentuu. Rakenne, josta voidaan käyttää myös mm. printtimediasta tuttua termiä taitto on sovelluksen käyttöliittymä pohja. Se määrittää, miten eri elementit asemoidaan käyttöliittymään. Yksittäiset käyttöliittymän elementit upotetaan rakenteeseen, ja ne ovat ohjelman rajapinta palvelun toiminnallisuuteen eli yhteys liiketoimintalogiikkaan käyttöliittymän takana. Visuaalinen ulkoasu on verho rakenteen ja elementtien päällä ja määrittää yksityiskohtaisesti miltä yksittäinen elementti näyttää käyttäjälle. (Sinkkonen ym. 2009: 215.)

Sisällön määritelmä on hieman abstraktimpi. Se on varsinainen anti, jonka käyttäjä saa vieraillessaan verkkosivulla tai käyttäessään siellä olevaa palvelua tai sovellusta. Se on usein informaatiota muodossa tai toisessa. Usein kaupallisissa toteutuksissa se koostuu siitä, mitä liiketoimintalogiikka antaa käyttäjälle tämän käyttäessä palvelua. Tyyliltään

se voi olla staattista, kuten erilaiset tekstijulkaisut, tai dynaamista, interaktiivisuutta vaativaa toiminnallisuutta. Sisältö pääsääntöisesti määrää sen miten käyttöliittymä tulisi rakentaa. Rakenne, yksittäiset elementit ja visuaalinen ulkoasu ovat avustavia ominaisuuksia ja toiminnallisuutta, joilla pyritään helpottamaan käyttäjää sisällön selaamisessa, tulkinnassa ja mielekkäässä käyttämisessä. Rakennetta, elementtejä ja visuaalisuutta manipuloimalla luodaan käyttöliittymä, johon sisältö sopii vaivattomasti, ja jossa käyttäjä kokeen itse sisällön käyttökokemuksen hyväksi. (Sinkkonen ym. 2009: 256.)

Käyttöliittymän toteutukseen ei ole valmiita malleja ole saatavilla, koska järkevä ja looginen toteutus riippuu paljon kohderyhmästä ja verkkosivun tarjoamasta sisällöstä. Yksinkertaisuus sivun rakenteessa ja sivustolla navigoitaessa on kuitenkin yksi selkeä perusvaatimus, joka kannattaa käyttöliittymälle asettaa. Tarjottavien palveluiden määrä kannattaa myös rajata kyseisen kohderyhmän osalta oleellisimpiin palveluihin, tai kaupallisessa toteutuksessa luomalla useampia moduuleita, joita käyttäjä voi halutessaan ottaa käyttöön. Näin säästytään siltä, ettei palvelun todellinen luonne katoa tukipalveluiden alle. Visuaalinen ilme on myös hyvin henkilökohtainen makukysymys. Hillityillä värivalinnoilla voi tosin päästä varmempaan lopputulokseen toisin kuin äärivalinnoilla.

Jatkomenettelyt

Kun liiketoimintalogiikka on valmis ja sille on rakennettu selkeä ja helppokäyttöinen käyttöliittymä, alkaa tuotteen ylläpito toimenpiteet. Vaikka ohjelmistototeutus olisikin tehty hyvin tarkasti ja ammattimaisesti eri menetelmiä hyödyntäen, on mahdollista, että joitakin odottamattomia käyttötapauksia tai kooditason virheitä ilmenee ja tuotetta pitää parantaa vastaamaan laatuvaatimuksia. Parhaassa tapauksessa odottamattomiin virheisiin on osattu varautua jo kooditasolla ja oireet eivät näy käyttäjälle asti, vaan virheilmoitukset tallentuvat lokiin. Näin saadaan pidettyä tuote käyttökelpoisena pienistä virheistä huolimatta ja kehitystiimi voi ratkoa ohjelmaan liittyviä ongelmia käyttäjän näkymättömissä. Helppo virheiden raportointi onkin oleellinen osa tehokasta ylläpitoa ja nopeuttaa itse prosessia, kun käyttäjät voidaan valjastaa löytämään tuotteissa mahdollisesti piileviä virheitä.

Ylläpito toimenpiteiden lisäksi saattavat alkaa myös jatkokehitystyöt, jos tuotteessa nähdään potentiaalia, ja uusia lisäarvoa tuottavia ominaisuuksia on mahdollista lisätä

valmistuneeseen toteutukseen. Pilvipalveluissa jatkokehitys on hyvin luontainen osa kehitysprosessia, koska markkinat ovat suuret ja kilpailu on kovaa. Räätälöityjen toteutusten jatkokehityksestä päättää asiakas, tai siitä on voitu tehdä mainintoja jo tuotteen tilausvaiheessa.

2.4 Ohjelmointikielet

Ohjelmointikieliä on suuri valikoima, Bille Kinnersleyn ylläpitämän kieli luettelon mukaan noin 2500 raportoitua (The Language List 2011), ja niillä kaikilla on omat erityispiirteensä. Kielet voidaan jakaa ryhmiin eri ominaispiirteiden mukaan. Osa on tehty hyvin kevyiksi ja yksinkertaisiksi, ja niillä toiminnallisuutta on mahdollista rakentaa nopeasti ja vaivattomasti. Nämä kielet ovat usein ns. skript-kieliä, joilla voidaan toteuttaa suuriakin ohjelmia, mutta jotka eivät välttämättä ole siihen ideaali valinta, esim. löysän määrittelyn tai muun rakenteellisen eroavaisuuden vuoksi. On myös raskaampia, monimutkaisempia ohjelmointikieliä, joita käytetään esimerkiksi käyttöjärjestelmien rakentamiseen. Sellaisia käytetään tehtävissä, joissa vaaditaan tarkkoja määritelmiä, paljon eri ominaisuuksia ja käytettävältä kieleltä vaaditaan tehokkuutta.

Kieliä on suunniteltu eri alustoille ja toimimaan erilaisissa ympäristöissä. Monissa on paljon samaa, rakenteessa tai syntaksissa, koska ne polveutuvat esim. samasta kielipuusta. Tämä mahdollistaa sen, että useimpiin tehtäviin on olemassa useampikin kieli, joka soveltuu vaadittavan toiminnallisuuden toteuttamiseen erityisen hyvin. Ohjelmistotuotannossa on kuitenkin vakiintunut selkeästi joukko kieliä, joiden kehitys on aktiivisempaa ja joita käytetään useimmin kaupallisissa toteutuksissa. Mm. Java- ja JavaScript-kieltä käytetään laajalti pilvipalveluiden toteuttamisessa, ja ne ovat saavuttaneet vankan aseman ohjelmistokielien standardien joukossa.

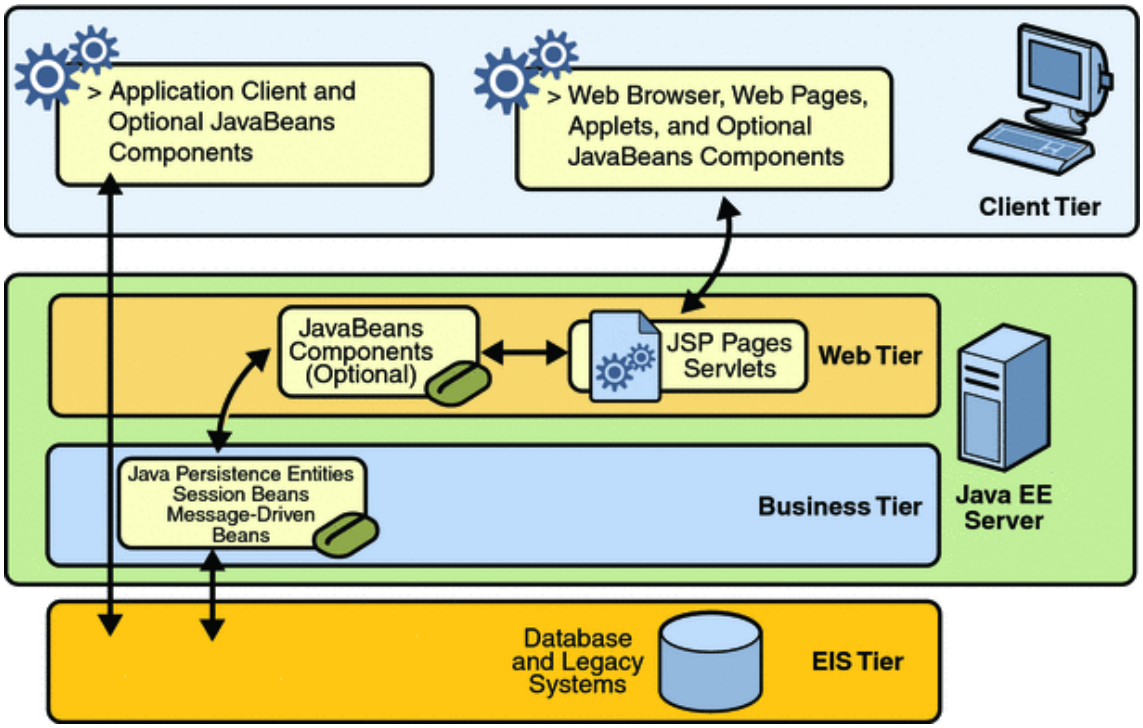
2.4.1 Java-ohjelmointikieli ja Java EE 5 -kehitysalusta

Java on luokkarakenteeseen perustuva olio-ohjelmointikieli, jonka alun perin kehitti Sun Corporation. Java on erittäin monipuolinen kieli. Sen ominaispiirteisiin kuuluu alustariippumattomuus, eli se käyttää samaa teknologiaa asiakas- ja palvelin-sovelluksissa. Tämä mahdollistaa turvallisen ja tehokkaan tiedonvaihdon asiakaspäätteen ja palvelimen välillä. Javalle on kolme eri kehitysalustaa, Micro Edition, Standard Edition ja Enterprise Edition. Kehitysalustasta riippuen ne sisältävät eri kokoelman työkaluja ja ominaisuuksia, ja ne ovat tarkoitettu hyödynnettäviksi sen

mukaan eri kohdeympäristössä. Micro Edition on suunnattu mobiililaitteille, Enterprise Edition yritys- ja palvelinkäyttöön, ja Standard Edition niiden väliin ns. yleisversioksi kotitietokoneelle. Java tarjoaa yhdenmukaisen ohjelmointimallin, jota voi käyttää erilaisissa ympäristöissä ja eri toiminnallisuutta vaativissa tehtävissä erittäin tehokkaasti. (Distributed Multitiered Applications - The Java EE 5 Tutorial 2010.)

Java Enterprise Edition tarjoaa erittäin laajan ja kattavan sarjan ohjelmointirajapintoja, ja työkaluja sovelluskehittäjille nopeampaan, joustavampaan ja luotettavampaan sekä turvallisempaan ohjelmistokehitykseen. Kehitysalustan laajuuden vuoksi luku ei paneudu teknillisiin yksityiskohtiin, vaan annetaan yleiskäsitys Java EE -ohjelmistojen rakenteesta. Lukija voi halutessaan perehtyä Java EE -kehitysalustaan tarkemmin lähteissä olevan Oraclen tarjoaman Java EE 5 -johdatuskurssin kautta.

Java EE -alusta käyttää jaettua monikerroksista ohjelmistomallia, jossa kaikki yrityssovelluksen komponentit on jaettu tehtävänsä luonteen mukaan eri kerroksille. Kerrokset ovat Client, Web, Business ja EIS Tier. Vapaasti suomennettuna nämä ovat asiakas-, Internet-, yritys- ja tietokantakerros. Eri kerroksilla olevat komponentit ajetaan myös omissa laitteissaan kyseisen kerroksen mukaan. Asiakaskerroksen komponentit suoritetaan asiakkaan päätelaitteella, Internet- ja yrityskerros ajetaan Java EE -palvelimella ja tietokantakerroksen komponentit suoritetaan omalla EIS palvelimellaan. Fyysisesti Java EE -palvelimen ja EIS-palvelimen ei tarvitse sijaita eri laitteilla, mutta suuremmissa organisaatioissa ne on tapana eriyttää omiksi yksiköikseen. (Distributed Multitiered Applications - The Java EE 5 Tutorial 2010.) Alla kuviossa 5 havainnollistetaan jaettujen monikerroksellisten ohjelmistojen rakennetta ja komponentien välisiä suhteita.



Kuvio 5. Jaetun monikerroksisen Java EE -ohjelman rakenne ja komponenttien väliset suhteet (Distributed Multitiered Applications - The Java EE 5 Tutorial 2010).

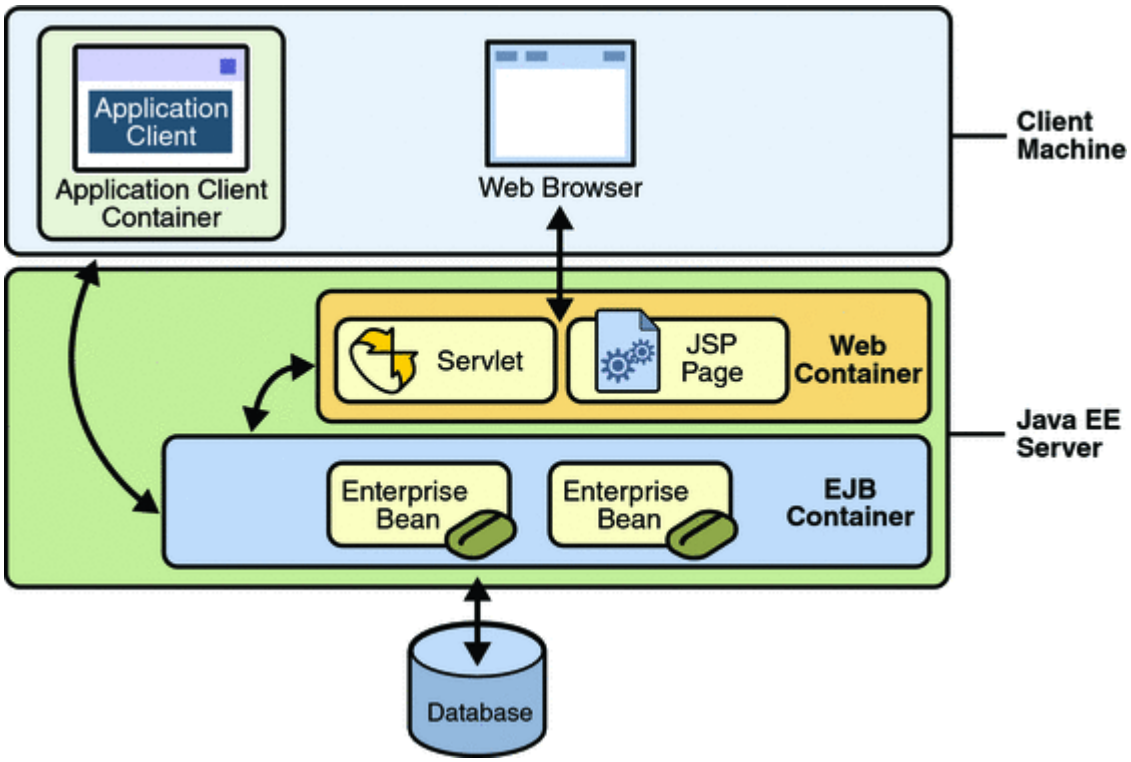
Java EE -ohjelmat koostuvat yksittäisistä komponenteista. Jokainen komponentti on oma toimiva ohjelmayksikkönsä, ja kykenevä ajettavaksi itsekseen. Komponentteja ovat asiakaskomponentit ja Java Appletit, jotka suoritetaan asiakaskerroksessa. Internet-kerroksen komponentit ovat JSP (JavaServer Pages), JSF (JavaServer Faces) ja servlet-komponentit, jotka ajetaan Java EE -palvelimella, ja yrityskerroksen Enterprise JavaBean-komponentit, jotka ajetaan myös Java EE -palvelimella. (Distributed Multitiered Applications - The Java EE 5 Tutorial 2010.)

Asiakaskomponentit voidaan jakaa web- ja asiakasohjelma-komponentteihin. Internet-asiakaskomponentti on dynaamista tai staattista sisältöä sisältävä verkkosivu, jonka selain hakee palvelimelta ja näyttää selaimen kautta asiakkaalle. Internet-komponentteja sanotaan tyypillisesti kevyiksi asiakkaiksi, koska ne eivät yleensä suorita raskaita operaatioita ja toimivat selaimen kautta. Java EE -palvelin ja yrityskerros suorittavat monimutkaiset operaatiot ja Internet-asiakkaat keskittyvät saamansa tiedon näyttämiseen. Verkkosivu saattaa sisältää myös Java Applet-komponentteja, jotka ovat verkkosivuun upotettuja, Java-virtuaalikoneen kautta suoritettavia ohjelmia. Ohjelmat ajetaan Java-virtuaalikoneen kautta, kun asiakas lataa verkkosivun, ja näytetään asiakkaalle normaalissa selainnäkymässä. Java Appletit vaativat Java-

virtuaalikoneliitännäisen asennettuna selaimeen toimiakseen oikein. (Distributed Multitiered Applications - The Java EE 5 Tutorial 2010.)

Asiakasohjelmat taas ovat täydellisiä Java-ohjelmia ja ne suoritetaan muiden tietokoneohjelmien tapaan käyttöjärjestelmä ympäristössä. Ne ovat paljon monipuolisempia ja raskaampia asiakkaita verrattuna Internet-asiakkaisiin ja tarjoavat yleensä enemmän toiminnallisuutta ja ominaisuuksia, koska voivat suorittaa yritys-kerroksen operaatioita. Ohjelmalla voi yleensä muodostaa yhteyden Java EE -palvelimeen ja sitä kautta hyödyntää myös tietokantapalvelinta ja suorittaa monimutkaisia operaatioita. (Distributed Multitiered Applications - The Java EE 5 Tutorial 2010.)

Verkkosivuja muodostavat Internet-asiakaskomponentit voivat olla joko servlettejä tai JSP-sivut, ja niissä on voitu hyödyntää JSF-teknologiaa. Servletit ovat Java-ohjelmointikielellä tehtyjä luokkia, jotka prosessoivat saamansa pyynnön tai kutsun selaimelta ja muodostavat palvelimella pyynnön pohjalta HTML-pohjaisen vastauksen, jonka selain kykenee tulkitsemaan ja näyttämään. JSP-sivut ovat enemmän tekstipohjaisia, staattisia HTML-sivuja rakenteeltaan muistuttavia Java- ja HTML-koodia sisältäviä verkkosivuja. Ne prosessoidaan Java EE -palvelimella ja lähettävät vastauksen kuten servletitkin, mutta antavat kehittäjille tavanomaisten staattisten HTML-sivujen kehitysnäkymän, ja mahdollisuuden luoda vaivattomasti verkkosivujen staattinen osuus. JSF-teknologia yhdistää JSP-sivut ja servletit, ja tarjoaa käyttöliittymä komponenttikehyksen verkkopalveluiden kehitystä varten. Tämä on laajemmin dokumentoituna ja käytössä Java EE 6 -versiossa. Alla kuviossa 6 hahmotellaan tarkemmin komponenttien säiliöiden välisiä suhteita. (Distributed Multitiered Applications - The Java EE 5 Tutorial 2010.)



Kuvio 6. Java EE -säiliöiden väliset suhteet (Distributed Multitiered Applications - The Java EE 5 Tutorial 2010).

Useassa eri kerroksessa toimivat komponentit toimivat suhteessa toistensa kanssa, ja nämä komponenttien väliset suhteet pitää hoitaa tehokkaasti, jotta ohjelma voi toimia turvallisesti ja luotettavasti. Java EE:ssä tämä on ratkaistu käyttämällä säiliöitä, joihin eri komponentit sijoitetaan. Ennen komponenttien suorittamista, niistä pitää luoda moduuleita, jotka ladataan tiettyyn säiliöön. Säiliölle määritellyt ominaisuudet määrittävät sen sisällä olevien yksittäisten komponenttien käyttäytymisen, ja itse komponenttien muodostaman ohjelman käyttäytymisen Java EE -palvelimella. Komponenteille voidaan siis määrittää erilaisia ominaisuuksia, ja niiden käytöstä voidaan ohjata riippuen siitä, missä säiliössä ne operoivat. (Java EE Containers – The Java EE 5 Tutorial 2010.)

2.4.2 Web-teknologiat

XHTML ja HTML DOM

HTML eli Hypertext Markup Language on julkaisu- ja merkintäkieli, jota käytetään verkkosivujen luomiseen. HTML mahdollistaa tekstin, erillisten dokumenttien, kuvien, lomakkeiden jne. julkaisemisen Internetissä halutussa muodossa ja tyyleillä varustettuna. Kieli syntyi tarpeesta saada globaali, standardoitu kieli, jolla kaikki

voisivat julkaista tietoa julkisessa jakelussa, ja mitä kaikki päätelaitteet voisivat potentiaalisesti ymmärtää. HTML-kielen kehitti alun perin Tim Berners-Lee 90-luvulla, ja kielen kehitys on ollut aktiivista siitä lähtien. Tällä hetkellä yleisessä käytössä ovat rinnakkain HTML 4.0/4.1 ja HTML5-standardit, ja niiden laajennukset XHTML 1.0 ja XHTML5, missä X tarkoittaa eXtended. Alla käsitellään tarkemmin HTML 4.1 standardia sen laajemman levinneisyyden ja vakiintuneisuuden vuoksi. (HTML 4.01 Specification 1999.)

HTML-kielen perusosia ovat elementit, joilla luodaan sivun rakenne ja ulkoasu. Elementit sisältävät aloitusmerkin ja yleensä myös erillisen lopetusmerkin, joiden avulla määritellään elementin sijainti dokumentissa suhteessa muihin elementteihin. Eri elementeillä on oma tehtävänsä, ja ne palvelevat sen mukaista tarkoitusta. HTML-sivun ns. juurielementti eli elementti, jonka sisälle kaikki muut elementit asettuvat, on HTML-elementti. Sivu jakautuu HTML-elementin sisällä kahteen osaa, päähän (head-elementti) ja runkoon (body-elementti). Näillä kahdella pääosalla on selkeä ero, joka vaikuttaa näytettävän dokumentin prosessointiin, ulkoasuun ja rakenteeseen. Selaimen ladatessa HTML-sivua, se prosessoi head-elementin ja sen sisällön, ennen kuin mitään näytetään käyttäjälle selaimessa, ja body-elementti latautuu näytön yhteydessä. Tämä määrittelee oleellisesti sen, mitä kummankin elementin sisälle tulisi sisällyttää. Pääsääntöisesti head-elementin sisälle tulisi laittaa tietoa kyseessä olevasta dokumentista, kuten tyylimäärittelyt ja toiminnallisuutta tuovat skriptit. Headin sisältö ei pääsääntöisesti tule näkyviin käyttäjälle sivua ladattaessa. Bodyn sisälle tulisi laittaa itse dokumentin sisältö, otsikot, teksti kappaleet, taulukot, kuvat yms. käyttäjälle näytettävää tietoa. Näiden lisäksi on suuri joukko muita, tarkemmin määriteltyjä elementtejä, joita voidaan käyttää verkkosivun luomiseen. (HTML 4.01 Specification 1999.)

Kaikilla elementeillä on tiettyjä yleisiä ominaisuuksia eli attribuutteja, ja mahdollisesti myös elementtikohtaisia ominaisuuksia, ja niillä tietty arvo. Ominaisuuksien avulla voidaan määritellä elementin sisältöä ja ulkoasua sekä yksilöidä niitä tai ryhmitellä nimi- tai luokka-ominaisuuden mukaan. Ominaisuus voi olla myös jokin ennalta määritelty tapahtuma, joka tulee laukaistuksi, kun käyttäjä tekee jotain verkkosivulla. Tapahtumat ottavat arvokseen tietyn JavaScript-funktion, joka suoritetaan tapahtuman ilmetessä Tämä mahdollistaa interaktiivisuuden luomisen verkkosivuille. HTML-sivuun

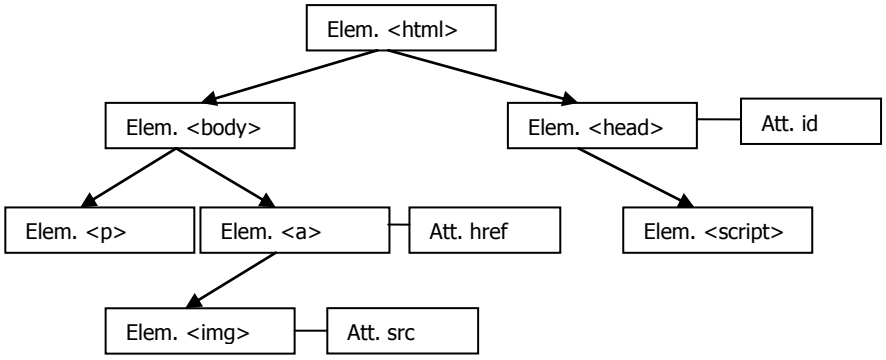
voidaan myös sisällyttää muiden ohjelmointikielien tapaan kommentteja, jotka eivät näy verkkosivujen käyttäjälle suoraan. (HTML 4.01 Specification 1999.)

DOM on lyhenne sanoista Document Object Model. DOM on ohjelmointirajapinta HTML- ja XML-tiedostoihin ja sen avulla ohjelmoijat voivat käsitellä, muokata, poistaa, jne. manipuloida ko. tiedostoja. Kaikki osat mitä validista HMTL tai XML dokumentista löytyy, voidaan käsitellä DOM:n avulla. DOM määrittelee tiedoston rakenteen, ja sen miten eri objekteja, kuten HTML-elementtejä, hallinnoidaan. HTML ja XML dokumenttien hierarkkisesta luonteesta johtuen, dokumentin rakenne on helppo esittää puu-mallina (Document Object Model (DOM) Level 3 Core Specification 2004.) Alla esimerkki HTML-dokumentin koodista (kuvio 7).

```
<html>
     <head id='id123'>
          <script>
               function test(){ alert('This is alert!'); }
          </script>
     </head>
     <body>
          <p>Kappale</p>
          <a href='www.example.com'>
               <img src='/kansio/kuva.jpeg' />
          </a>
     </body>
</html>
```

Kuvio 7. HTML-dokumentin esimerkki koodi

Kuvion 7 HTML-koodi on esitetty DOM-muodossa kuviossa 8.



Kuvio 8. HTML-dokumentti esimerkin DOM esitys

DOMissa dokumentilla on selkeä rakenne, joka voidaan esittää puurakenteena, joskin olisi ehkä mielekkäämpää puhua metsästä, koska usein dokumenteissa on useampia

puuhaarakkeita. DOMissa ei kuitenkaan määritellä, että dokumentin tieto tai objektien pitäisi implementoida tietyllä tavalla. DOMin työkalujen ja objektien avulla puussa voi liikkua elementistä toiseen ja takaisin, mikä helpottaa ohjelmoijien työtä hahmottaa tietorakennetta objektimallin avulla. (Document Object Model (DOM) Level 3 Core Specification. 2004.)

DOMin avulla HTML-dokumentin esitettävät elementit saatetaan ohjelmoijan ulottuville, ja niitä voidaan manipuloida esimerkiksi JavaScriptin avulla. Näin mahdollistetaan laajemmassa kontekstissa tunnettujen dynaamisten HTML-sivujen luonti.

JavaScript ja AJAX

JavaScript on ohjelmistotuotannossa laajalti käytetty ja tunnettu, Netscape-selainyhtiön kehittämä käyttäjäpuolen skript-ohjelmointikieli. Sitä on käytetty laajalti interaktiivisuuden ja dynaamisuuden luomisessa verkkosivuille. Ominaisuuksiltaan se on hyvin kevyt, pieni ja löysästi määritelty ohjelmointikieli, joka mahdollistaa sen monipuolisen ja joustavan käytön erilaista toiminnallisuutta luodessa. Se on kuitenkin rakenteellisesti ja määritelmiltään liian kevyt ja yksinkertainen käytettäväksi itsekseen ja vaatii aina ohjelmallisen käyttöympäristön, johon se on upotettu, kuten esimerkiksi verkkoselaimen. (JavaScript Guide 2011.)

JavaScript on oliopohjainen ohjelmointikieli. Oliopohjainen viittaa siihen, että kielessä määritellään tietyntyyppinen objekti, jolla on tietyt toiminnot ja ominaisuudet. JavaScriptissä on alustavasti määritelty joitakin tyypillisiä objekteja, jotka ovat tuttuja myös muista ohjelmointikielistä. Näitä ovat esimerkiksi taulut, listat, päivämäärä jne. Käyttäjällä on mahdollista myös itse luoda omia objekteja. Myös kaikki tavallisimmat ohjelmointikielten elementit, kuten perintä, loogiset operaattorit, kontrollirakenteet ja lausekkeet, ovat tarjolla myös JavaScript-kielessä. (JavaScript Guide 2011.)

JavaScript on saanut vaikutteita mm. Java-ohjelmointikielestä. Näillä kielillä on kuitenkin myös hyvin paljon eroja, ja ne toimivat eri periaatteiden mukaan osittain erilaisissa ympäristöissä. Java-kielelle on tyypillistä staattiset tyyppimäärittelyt objekteja luotaessa. Tämä tarkoittaa, että ohjelmoijan pitää ennalta tietää, minkälaista objektia on luomassa. Java on luokka-pohjainen ohjelmointikieli kun taas JavaScript on prototyyppimalliin perustuva kieli. Sille on ominaista löysät objektin tyyppimäärittelyt, jossa objektin tai muuttujan ei määritellä ollenkaan, vaan kielen kääntäjä tekee sen

automaattisesti. Prototyyppimalli on myös vähemmän tiukka objektien hierarkkisten suhteiden hallinnassa. Se mahdollistaa dynaamisen perinnän riippuen objektin mahdollisuuksista verrattuna luokkapohjaiseen hyvin hierarkkiseen, staattiseen perintään. (JavaScript Guide 2011.)

Alustariippumattomana ohjelmointikielenä JavaScript toimii useimmilla eri selaimilla. Tämä mahdollistaa yhden ohjelmaversion tekemisen, joka toimii, pienellä varauksella, useimmissa eri selaimissa. Jotkut selaimet ovat määritelleet samaa toiminnallisuutta erilasisella syntaksilla, minkä johdosta tietyt elementit ja objektit ovat erinimisiä eri selaimilla. Pääsääntöisesti ainoastaan Internet Explorer -selaimeen on määritelty eri syntaksilla omia elementtejä, jotka eivät ole samanlaiset muilla selainvalmistajien tuotteilla. Muut selainvalmistajat tarjoavat yleensä standardoidumman yhteisen rajapinnan. Käyttäjälle tämä toiminnallisuuden elementtien eriävyys on kuitenkin pimennossa, mikä mahdollistaa yhtenevän käyttökokemuksen tarjoamisen, selaimesta riippumatta. (JavaScript Guide 2011.)

AJAX on lyhenne sanoista Asyncronous JavaScript and XML. AJAX ei ole ohjelmointikieli vaan on tekniikka, joka yhdistää JavaScript-kielen ja XML-tiedostojen käsittelyn. Tekniikkaa käytetään silloin, kun halutaan ladata verkkosivulle uutta sisältöä, ilman että koko sivua joudutaan päivitetään eli hakemaan uudestaan. AJAXia hyödyntämällä kehittäjän voivat pilkkoa verkkosivun osiin ja päivittää eri osat aina kun, tilanne niin vaatii. Tekniikka on erittäin hyödyllinen, kun ajatellaan suuria sivuja, joissa on paljon sisältöä ja yksittäin latauskerta vie suhteellisesti paljon aikaa. Jos sivut jouduttaisiin päivittämään uudestaan joka kerta, kun käyttäjä suorittaa tietyn toiminnon, voisi käyttökokemus osoittautua huonoksi. Käyttäjä joutuu odottamaan kohtuuttoman kauan staattisen sisällön uudelleenlatausta, mikä lopulta syö palvelun miellyttävyyttä käyttäjän näkökulmasta. AJAX-tekniikkaa hyödyntäen vain se osa, joka pitää päivittää haetaan uudestaan palvelimelta. Sisältö latautuu nopeammin, ja se mahdollistaa käyttäjän miellyttävämmän käyttökokemuksen.

3 Toteutus

3.1 Projektin alustus

Projektin suunnittelu pohjautuu kyseessä olevan liiketoimintaongelman ratkaisemiseen. Lopputuotoksena asiakkaille on tarkoitus tarjota uusi ominaisuus, jonka avulla he Genisysin tarjoamana CRM-palvelun käyttäjinä voivat tallentaa asiakkaidensa kanssa käytyjä sähköpostikeskusteluita CRM-tietokantaan. Käyttäjän lähettäessä viestejä asiakkailleen ohjataan nämä viestit lähetyksen yhteydessä myös Genisysin sähköpostipalvelimelle. Palvelin vastaanottaa viestit ja sähköpostikäsittelijäolio, joka on projektin liiketoiminnallisen logiikan lopputulos, prosessoi saamansa viestit ja syöttää sähköpostiviesteistä halutut tiedot tietokantaan. Tietojen tallentuessa tietokantaan käyttäjä pääsee tarkastelemaan eri asiakkailleen lähettämiään sähköpostiviestejä CRM-moduulin kautta. CRM-moduulissa on tällä hetkellä jo valmista toiminnallisuutta, jonka avulla eri aktiviteetteja käsitellään ja näytetään käyttäjälle, ja tätä toiminnallisuutta käytetään hyödyksi myös ko. projektin käyttöliittymäosuuden toteutuksessa.

Projektin kehitysalustaksi on järkevää ja luonnollista valita Java EE, koska se tarjoaa laajan ja kattavan kokoelman valmiita työkaluja verkkopalveluiden toteutukseen. Genisys hyödyntää Java EE -alustaa ennestään jo muissa pilvipalveluissaan, joten on luonnollista hyödyntää eri palveluiden yhteistä rajapintaa, jonka avulla toiminnallisuutta voidaan sitoa yhteen samaa teknologiaa hyödyntäen. Java on myös ohjelmointikielenä hyvin rikas ja monipuolinen tarjoten ominaisuuksia, jotka mahdollistavat vaativankin toiminnallisuuden toteuttamisen yksinkertaisesti ja vaivattomasti.

Projektin kannalta oleellisia, valmiita työkaluja ovat mm. Genisysin aiemman ohjelmistokehityksen tuotokset ja vapaasti hyödynnettävät Java-kirjastot, jotka ovat osa kattavaa Java EE APIa. JavaMail API on yksi Java EE APIn osa ja yksi projektin oleellisimmista ohjelmointirajapinnoista, koska se sisältää metodeita, joiden avulla muodostetaan yhteys sähköpostipalvelimelle ja käsitellään palvelimella olevia viestejä. Tietokantaoperaatiot suoritetaan hyödyntämällä Genisysin omaa tietokantaoperaatioihin perustuvaa Utils-luokkaa. Luokka sisältää kokoelman staattisia metodeita, joiden avulla erilaisia tietokantaoperaatioita voidaan suorittaa. Funktioille annetaan parametrina haluttu SQL-kysely, jonka se ohjaa tietokannalle suoritettavaksi. Näin säästytään erillisten tietokantaoperaatio funktioiden luomiselta.

Projekti voidaan jakaa kahteen päävaiheeseen, joista syntyy oma hallinnallinen inkrementtinsä, sovelluslogiikka, jonka avulla Genisysin sähköpostipalvelimelle saapuneet sähköpostit käsitellään ja prosessoidaan sellaiseen muotoon, että ne voidaan syöttää asiakkaan tietokantaan. Toinen inkrementti on itse CRM-moduulin käyttöliittymän muokkaaminen sellaiseen muotoon, että se kykenee näyttämään sähköpostiviestejä selkeästi ja yksinkertaisesti. Nämä osat muodostavat lopullisen tuotoksen, toimivan sovelluksen, joka on projektin päämäärä.

Projektissa noudatetaan ketterän kehityksen ohjelmistotuotannon periaatteita. Scrum-mallia hyödynnetään projektin sujuvan etenemisen varmistamiseksi. Mallia ei voida hyödyntää täysin, koska ennalta määritellyt roolit tulisivat olemaan ehkä liian päällekkäisiä. Voidaan nähdä, että asiakkaan roolia toimittaisi yrityksen toimitusjohtaja, joka toimisi ideallisesti Scrum-mestarin roolissa, kehitystiimin koostuessa yhdestä henkilöstä. Mutta yksilötasolla projektin kannalta oleellisia periaatteita noudattamalla voidaan pyrkiä saavuttaamaan lopputulos, johon on päästy hallitusti ja systemaattisen kehitystyön tuloksena. Oleellisina olisi nostaa esille päämääränä olevan toimiva sovellus, johon on pyrittävä pääsemään mahdollisimman yksinkertaisella toteutuksella kuitenkaan joustamatta sovellukselle annetuista vaatimuksista. Projektissa ei ole määrittelty alemman ohjelmointitason menetelmiä erikseen. Kehitystiimin koostuessa vain yhdestä henkilöstä ei ole mielekästä käyttää aikaa ohjelmointitapojen tarkkaa määrittelyyn. Scrum määrittelee ylemmän tason menetelmät ja tavoitteet, joita pidetään riittävinä projektin tehokkaan loppuun viemisen kannalta.

3.2 Määrittely ja suunnittelu

Sähköpostikäsittelijälle asetetaan tiettyjä vaatimuksia, jotka sen on täytettävä, jotta projektin päätteeksi voidaan analysoida lopputuotoksen oikeellisuutta. Uutta ominaisuutta hyödyksi käyttäen käyttäjän pitää olla mahdollista saada viestit talteen omaan tietokantaansa, ja hänen on oltava mahdollista viestiä käyttämällä erilaisia sähköpostiformaatteja. Erilaisia formaatteja sähköpostiviestinnässä ovat puhtaaseen tekstiin perustuvat viestit, HTML-kieleen perustuvat viestit, joista suurin osa viesteistä koostuu. Microsoftin tarjoamassa Outlook-sähköpostiohjelmassa on tarjolla myös RTF-tiedostomuoto. RTF-tiedostomuodon harvan käytettävyyden vuoksi itse tekstin formatointia ei tueta, mutta sisällön teksti prosessoidaan puhtaana tekstinä ilman formatointeja. Sähköpostin lähettäjän osoite eli CRM-palvelun käyttäjän sähköpostiosoite tulee myös olla merkittynä ko. henkilön sähköpostiosoitteeksi CRM-

palvelun käyttäjä tietoihin. Viestit prosessoidaan sähköpostikohtaisesti, ja jos viestejä lähetetään tunnistamattomista osoitteista, joita ei ole määritelty CRM-palveluun, niitä ei myöskään alustavasti käsitellä. Sama koskee käyttäjän asiakaskohtaisia sähköpostiosoitteita.

Sähköpostiviestien prosessoimista varten luodaan oma olio eli objekti. Objekti on vastuussa viestien hakemisesta sähköpostipalvelimelta, niiden käsittelystä ja syöttämisestä tietokantaan. Sähköpostikäsittelijää voidaan pitää selkeänä käsitteenä, ja se on kokonaisuus, jolla on tiettyä toiminnallisuutta ja tiettyjä ominaisuuksia. Sähköpostikäsittelijää varten luodaan oma luokka nimeltä EmailHandler, joka sisältää nämä ominaisuudet ja funktiot. Sähköpostien käsittelijään liittyvät sähköpostiviestit, joten on järkevää hahmottaa mahdollisia viestityyppejä ja tunnistaa viesteihin liittyviä ominaisuuksia ennen kuin itse metodeita ruvetaan toteuttamaan. Sähköpostiviestin ulkoinen rakenne itsessään noudattaa standardien mukaista kaavaa, mutta viestin sisältö voi olla hyvin erilainen riippuen sähköposti-ohjelmasta, jolla viesti on luotu ja lähetetty. Käyttäjällä itsessään on mahdollisuus määritteellä erilaista sisältöä sähköpostiin, mikä pitää ottaa huomioon viestiä purettaessa, jotta viestin sisältö saadaan näytettyä alkuperäisessä muodossa.

Sähköpostiviesti koostuu osista, joista oleellisimmat tässä yhteydessä ovat viestin tunnistetiedot ja viestin sisältö. Sähköpostiviesti voi pitää sisällään erilaisia komponentteja kuten tekstiä, tekstin formatointiin ja asemointiin liittyvää tietoa, tekstin sekaan upotettuja kuvia ja erillisiä liitteitä. Tunnistetiedot määrittelevät mm. sen, minkälainen viesti on kyseessä, kuka sen lähetti ja kenelle kaikille se oli lähetetty. Tunnistetietojen avulla kaikki sähköpostiviestit ohjataan oikean käyttäjän CRM-tietokantaan.

Sähköpostiviesteille ja niiden sisältämille komponenteille on määritelty kullekin oma MIME-tyyppi, jonka avulla erityyppiset viestit voidaan erotella toisistaan. MIME tarkoittaa Multipurpose Internet Mail Extension eli vapaasti suomennettuna monikäyttöinen sähköpostiviesti laajennus. MIME-tyypeillä määritellään tunnistetietoihin sähköpostiviestien tyyppi, joka edelleen antaa tietoa viestin sisällöstä. Viestityyppien avulla mm. sähköpostiohjelmat tietävät, minkälainen viesti on kyseessä ja mitä tietoa sen pitäisi sisältää. Tämä helpottaa oleellisesti viestin sisällön purkamista

osiin. Näin ollen myös sähköpostikäsittelijän tulee osata tunnistaa viestin sisältämät eri komponentit, ja osata käsitellä viestit niiden sisällön vaatimalla tavalla.

Sähköpostiviestit ovat MIME-tyypeillä tulkittuna joko säiliö useammalle objektille yhdessä sähköpostiviestissä tai puhdasta tekstisisältöä oleva viesti. Sähköpostiohjelma rakentaa viestistä säiliön, jos siihen liitetään tekstin lisäksi muita objekteja. Liitetiedostoja sisältävät viestit ovat MIME-tyyppiltään multipart/mixed. Tämä kuvastaa sitä, että viestin sisältö voi olla hyvinkin sekalainen ja sisältää myös muita viestityyppejä, kuten toisia säiliö-objekteja. Sisällytettyjä kuvia sisältävät sähköpostiviestit ovat ominaispiirteiltään säiliöitä ja MIME-tyypiltään multipart/related. Samoin ovat vaihtoehtoista sisältöä sisältävät eli sekä teksti- ja HMTL-sisällöllä varustetut viestit, ovat tyypiltään multipart/alternative. Koska viestejä vastaanotettaessa ei tiedetä, minkälaisia viestejä joudutaan mahdollisesti käsittelemään, pitää eri tapauksia varten luoda oma käsittelytapa. Käytännössä katsoen toteutusta kehitetään tyyppi kerrallaan. Halutuille eri viestityypeille rakennetaan tuki, mutta varaudutaan myös siihen, että odottamattomia ja ehkä vähän harvinaisempia viesti-tyyppejä saada käsiteltyä onnistuneesti.

Sähköpostinkäsittelijäolio itsessään on manuaalisesti operoitava olio, joka vaatii tässä palvelussa erillisen kutsun, jonka jälkeen sähköpostien haku prosessi käynnistyy. Tätä tarkoitusta varten käytämme hyödyksi avoimeen lähdekoodin pohjautuvaa Quartz Scheduler -palvelua. Palvelun avulla saamme ajastettua sähköpostikäsittelijäolion kutsun haluamamme aikajakson välein. Näin saamme automatisoitua sähköpostien käsittelyn taustaprosessiksi, kun palvelun on todettu toimivan ennalta määriteltyjen tavoitteiden mukaan.

3.3 Toteutus ja testaus

Toteutus osuus jakautuu kahteen osaan, sovelluslogiikan eli liiketoimintalogiikan toteuttamiseen ja käyttöliittymätoteutukseen. Sovelluslogiikan alla pyritään kuvastamaan, miten palvelu toimii, ja käyttöliittymässä näytetään, miten liiketoimintalogiikan toiminnallisuus näkyy palvelua käyttävälle asiakkaalle. Liiketoimintalogiikan ohjelmakoodin on erillisellä liitteellä ja sovelluslogiikka luvussa käsiteltävät eri ominaisuudet ja funktiot viittaavat kyseiseen dokumenttiin.

3.3.1 Sovelluslogiikka

Sähköpostikäsittelijä-oliolla on tiettyjä ominaisuuksia ja ulkoisia palveluita, joita olio voi kutsua, sekä sisäisiä metodeita, jotka ovat ulkoisten palvelukutsujen tukitoimista. Funktiot jaotellaan kooditasolla yksityisiin eli sisäisiin ja julkisiin eli ulkoisiin palveluihin. Alla liiketoimintalogiikka käydään läpi niin kuin se periaatteessa ohjelman suorituksen aikana tulisi tapahtumaan.

Sähköpostikäsittelijä-olion toimintoihin kuuluu alustusfunktio eli konstruktori, joka luo uuden olion tietyillä ominaisuuksien arvoilla. Sähköpostikäsittelijä olion ominaisuuksiin kuuluu aina tietty sähköpostipalvelin, johon se ottaa yhteyttä ja mistä se tarkistaa, onko uusia viestejä tullut. Käsitteenä palvelimella on omat ominaisuutensa, jotka määritellään sähköpostikäsittelijän ominaisuuksiksi. Nämä ominaisuudet ovat yhteysosoite, käyttäjänimi, salasana ja palvelimen tyyppi. Lisäksi sähköpostikäsittelijä-oliolle on määritelty tietokanta, josta edellä mainitut tiedot haetaan.

Sähköpostipalvelimella on tietty yhteysosoite eli IP-osoite, jonka avulla se löydetään Internetistä. Palvelin vaatii myös tietyn käyttäjänimen ja salasanan, jotta saapuneisiin viesteihin päästäisiin käsiksi. Viestejä käsitellään tiettyä protokollaa hyväksi käyttäen ja se annetaan sähköpostikäsittelijälle palvelimen tyyppinä. Tulevia viestejä käsiteltäessä on kaksi eri protokolla vaihtoehtoa, POP ja IMAP. Tässä tapauksessa suosimme IMAP-protokollaa, koska se mahdollistaa viestien monipuolisemman käsittelyn verrattuna POP-protokollaan. Edellä mainitut ominaisuudet haetaan tietystä ns. ohjelmatietokannasta, joka on myös määritelty itse Geniservice-tietokantaan. Geniservice-tietokanta on vastuussa sähköpostipalvelimelle tulevista operaatioista, ja mm. eri Quartz-ajastukset ja niiden määrittelyt löytyvät sieltä omasta taulustaan.

Sähköpostikäsittelijä-oliota luotaessa konstruktorille ei syötetä mitään tietoa parametreina, vaan se kutsuu loadSettings-funktiota, joka suorittaa muutaman ennalta määritellyn tietokantakyselyn. Kyselyistä saatujen tulosten pohjalta syötetään sähköpostikäsittelijäolion ominaisuuksien arvoiksi tuloksista saadut arvot. Dynaamisesti hakemalla olion ominaisuuksien arvot kehityksestä vastuussa olevien henkilöiden ei tarvitse muuttaa toistuvasti olion-ominaisuuksien arvoja vastaamaan kehitysympäristön testiarvoja ja palvelimelle julkaistaessa virallisia käyttöarvoja.

Sovelluslogiikka perustuu neljää eri funktioon, joiden avulla itse sähköpostiviestien haku ja käsittely prosessi käydään läpi. Yksi funktio nimeltä checkMessages on ainoa

julkinen palvelu, joka oliolla on. Funktiot handleMessages, handleMessageContent ja saveFile ovat yksityisiä palveluita, joita kutsutaan tietyissä kohdissa ohjelmakoodia. Tämä funktio toimii ulkoisena rajapintana olion toiminnallisuuden käytölle. Koska sähköpostikäsittelijän funktio on hyvin yksinkertainen ja suoraviivainen, toteutushetkellä ei nähty tarvetta antaa muita toiminnallisuutta määriteleviä tai siihen vaikuttavia funktioita.

Uusi sähköpostikäsittelijä-olio luodaan ja funktioita checkMessages kutsutaan aina, kun ajastimen määrittelemien ehtojen mukainen tapahtuma ottaa paikkansa. Funktio checkMessages käyttää hyväkseen JavaMail APIn metodeita, joiden avulla se muodostaa uuden yhteyden olion ominaisuuksissa määriteltyjen sähköpostipalvelimen arvojen perusteella. Yhteyden muodostuttua funktio avaa käyttäjätunnuksen alla sijaitsevaan inbox-laatikkoon ja hakee sieltä kaikki viestit. Voidaan olettaa, että sähköpostikansio on tyhjä ja viestit uusia, koska funktio viestien haun jälkeen aina poistaa viestit. Jos uusia viestejä löytyi, funktio kutsuu sisäistä palvelua handleMessages, jolle se lähettää parametrina saamansa viestit JavaMail APIsta löytyvän Message-luokan objektina taulumuodossa. Jos viestejä ei löytynyt, funktio sulkee yhteyden sähköpostipalvelimeen ja fokus palaa luokkaan, josta funktiota kutsuttiin. Funktio herää uudestaan, kun Quartz-ajastus laukeaa ja uusi sähköpostikäsittelijä-olio luodaan ja funktiota kutsutaan.

Kun sähköpostikansiossa on viestejä, niin viestit käsitellään ja ensiksi viestien prosessoinnin aloittaa handleMessages-funktio. Funktio käy parametrina saamansa viestitaulun viestit yksi kerrallaan for-silmukan kautta läpi. Tässä funktiossa tarkastellaan ensin, löytyykö lähettäjä Genisysin tietokannasta, eli tarkastetaan onko viestin lähettäjä CRM-palvelun käyttäjä. Jos käyttäjää ei tunnisteta, viestiä ei prosessoida. Näin eliminoidaan virheellisesti sähköpostilaatikkoon eksyneet viestit. Tämän jälkeen tarkastellaan, löytyykö henkilö, jolle viesti on lähetetty, käyttäjän tietokannasta. Jos molemmat kyselyt tuottivat positiivisen tuloksen eli sekä CRM-palvelun käyttäjä on tunnistettu että käyttäjän asiakas on tunnistettu, voidaan viestin sisällön käsittely aloittaa. Jos viestin käyttäjää tai käyttäjän asiakasta ei tunnistettu, funktio hyppää viestin yli ja siirtyy seuraavaan.

Viestin sisällön käsittelee handleMessageContent-funktio. Aluksi käsiteltävälle viestille generoidaan yksilöllinen id-tunnus, jonka avulla se tunnistetaan ja linkitetään muihin

tietokenttiin. Tämä tapahtuu Genisysin oman Utils-luokan staattisen apufunktion avulla. Tämä jälkeen funktiossa on ryhmä valintalausekkeita, joiden avulla viestin sisältöä tunnistetaan ja käsitellään sisällön vaatimalla tavalla. Viestit jakaantuvat karkealla tasolla puhtaisiin teksti- tai HTML-pohjaisiin viesteihin, viesteihin, joissa on liitteitä, ja viesteihin, joissa on sisällytettyjä kuvia. Näistä muodostuu valintalausekkeen perusrakenne. Muitakin viestiformaatteja on olemassa, mutta niiden käyttö on harvinaista. Jos viestiä ei kuitenkaan saada kategorisoitua, siitä tulee ilmoitus järjestelmään ja viestityyppi tulostetaan lokiin. Valintalausekkeiden tarkoituksena on erotella tekstisisältö ja mahdolliset erilliset liitteet ja kuvat toisistaan. Kuvat ja liitteet tallennetaan asiakkaalle varattuun alikansioon Java EE -palvelimelta. Erilliset tiedostot linkitetään niille generoiduilla ohjaustiedoilla aina tiettyyn sähköpostiviestiin. Tiedostojen tallennuksen suorittaa saveFile-funktio, ja sitä kutsutaan aina, kun valintalausekkeesta päästään haaraan, jossa sisältö on tunnistettu olevan liitetiedosto tai sähköpostiviestiin upotettu kuva.

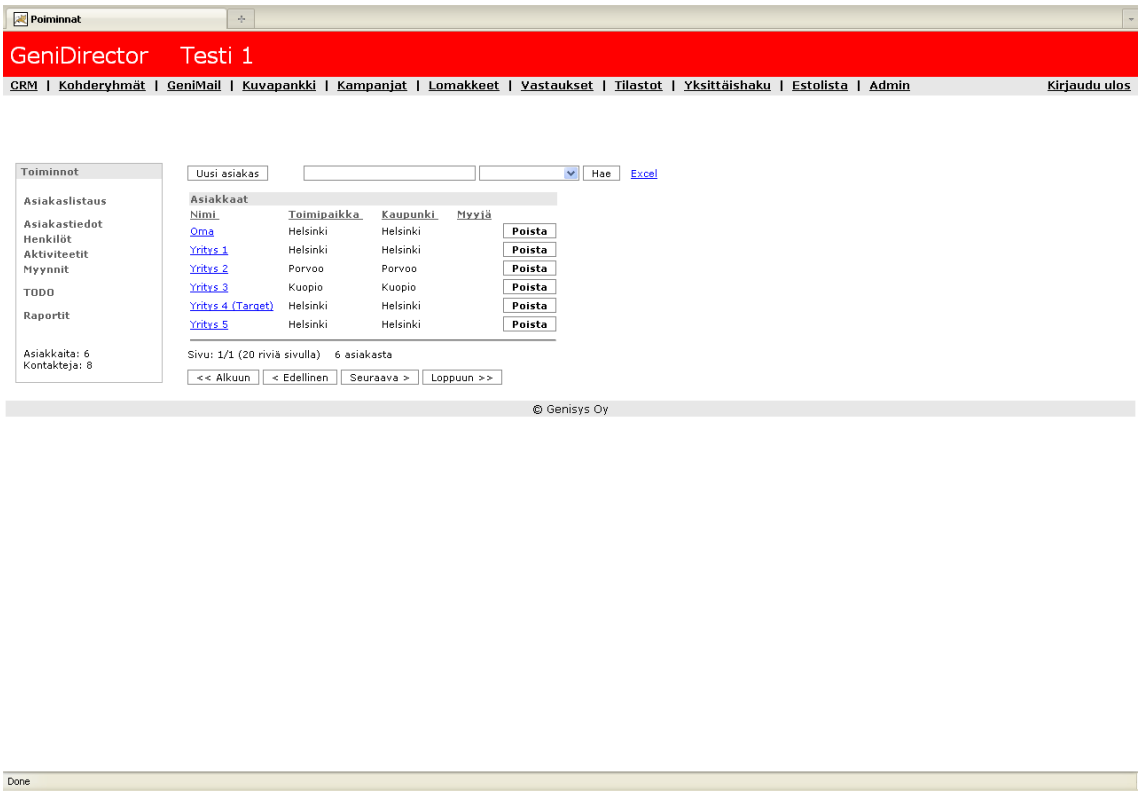
Kun viestin sisältö on lopulta käyty läpi, tallennetaan se käyttäjän tietokantaa aktiviteettitauluun, jossa ovat myös muut aktiviteetit. Tietokantakutsu tehdään hyödyntäen Genisysin Utils-luokan staattista tietokantaoperaatio funktiota databaseOperation. Funktiolle syötetään parametrina SQL-kysely, jonka se ohjaa tietokannalle prosessoitavaksi. Tässä tapauksessa kun sähköpostiviesti aktiviteettia ollaan syöttämässä tietokantaan, on kysely tyyppiä insert ja kyselyyn upotetaan muuttujina sähköpostista tallennusta varten haluttu tieto. Sähköpostiviestin sisällöstä karsitaan tietyt ei-halutut merkit, jotka voivat olla ristiriidassa SQL-kyselyn kanssa. Tähän hyödynnetään Utils-luokan funktiota escapeSQL, joka muokkaa sisältöä niin, että se ei aiheuta SQL-virhettä. Tietokantaan tallentuu mm. lähettäjän ja vastaanottaja osoitteet, viestin sisältö ja ohjaustietoa aktiviteetin tyypistä. Tässä tapauksessa aktiviteetti on tyyppiä sähköpostiviesti. Kun aktiviteetteja käsitellään käyttöliittymässä, ne näytetään sisällön vaatimalla tavalla ohjaustietojen avulla. Viestissä mahdollisesti mukana olleet liitteet tallentuivat saveFile-funktion sisällä olevalla omalla tietokantakutsullaan.

Koko viestitaulu käydään läpi viesti kerrallaan edellä mainitulla rutiinilla, kunnes kaikki viestin on käsitelty. Lopulta funktiot palaavat sinne, missä niitä kutsuttiin, ja lopulta päädytään takaisin checkMessages-funktioon. Lopuksi poistetaan kaikki viestit, jotka sähköpostilaatikossa oli, ja yhteys sähköpostipalvelimeen suljetaan. Ohjelman suoritus

loppuu tähän. Uusi sähköpostinkäsittelijä-olio luodaan ja checkMessages-funktiota kutsutaan, kun seuraava Quartz-ajastus laukeaa.

3.3.2 Käyttöliittymä

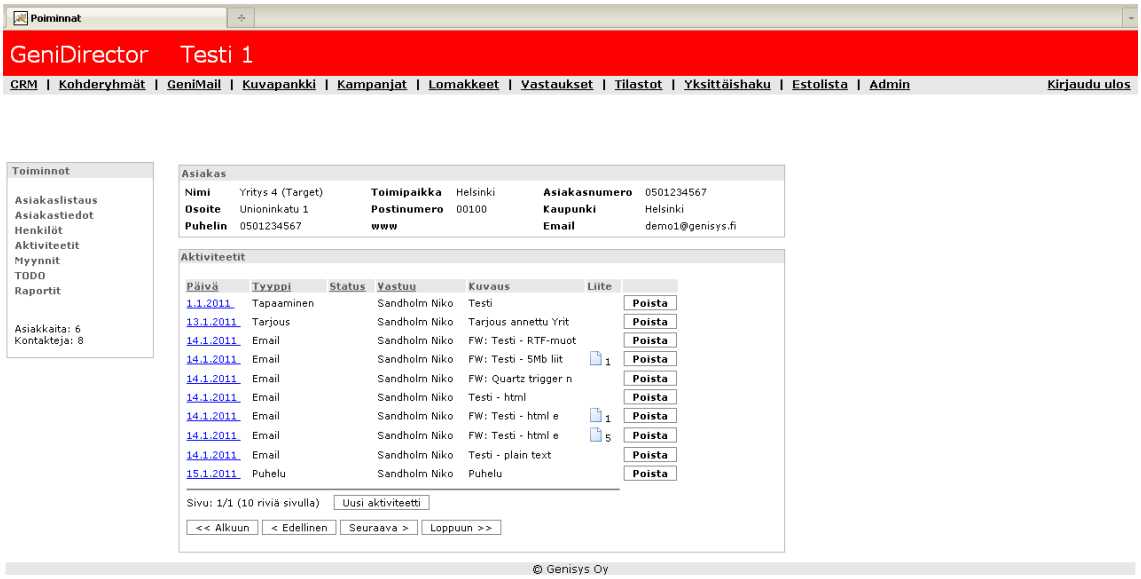
Käyttöliittymä luvussa tarkastelemaan sitä, miten toiminnallisuus tulee näkymään itse CRM-palvelun käyttöliittymässä. Palvelu ja sen käyttöliittymä on suunniteltu jo hyvin pitkälle, joten uutta toiminnallisuutta sisään tuotaessa on huomioitava olemassa olevat rajoitteet, jotka käyttöliittymälle on asetettu. Uutta toiminnallisuutta tuodessa palveluun pitää ottaa huomioon aiemmat ratkaisut ja sivuston rakenne ja pyrkiä upottamaan toiminnallisuus saumattomasti olemassa olevaan rakenteeseen. Alla kuvassa 1 käymme ensiksi läpi sen, miltä käyttöliittymä käyttäjälle näyttää ja mitä siinä tapahtuu, minkä jälkeen selostetaan toiminnallisuutta verkkosivujen ulkoasun takana.



Kuva 1. Genisysin GeniCRM-moduulin aloitusikkuna Internet-selaimessa.

GeniCRM-moduulin aloitus sivu on JSP-sivu, johon käyttäjä tulee halutessaan tarkastella asiakkaidensa tietoja. Kyseisessä näkymässä listataan palvelua käyttävän yrityksen asiakkaat taulukkonäkymään nimen, toimipaikan, kaupungin ja myyjän mukaan. Asiakkaat haetaan dynaamisesti JSP-sivua luotaessa Genisysin tietokantapalvelimelta asiakaan CRM-tietokannan alta. Vasemmassa sivussa ovat

palvelun navigointilinkit, joiden kautta asiakas pääsee tutkimaan asiakkaidensa tietoja, tarkastelemaan myyjien tallentamia myyntimahdollisuuksia, aktiviteetteja ylipäätään, tehtävälistaa tai raportteja. Käyttäjä valitsee ensin asiakkaan klikkaamalla sen nimeä listauksesta, jonka jälkeen hän pääsee tutkimaan asiakkaaseen liittyviä tietoja. All kuvassa 2 on navigoitu aktiviteetti näkymään.

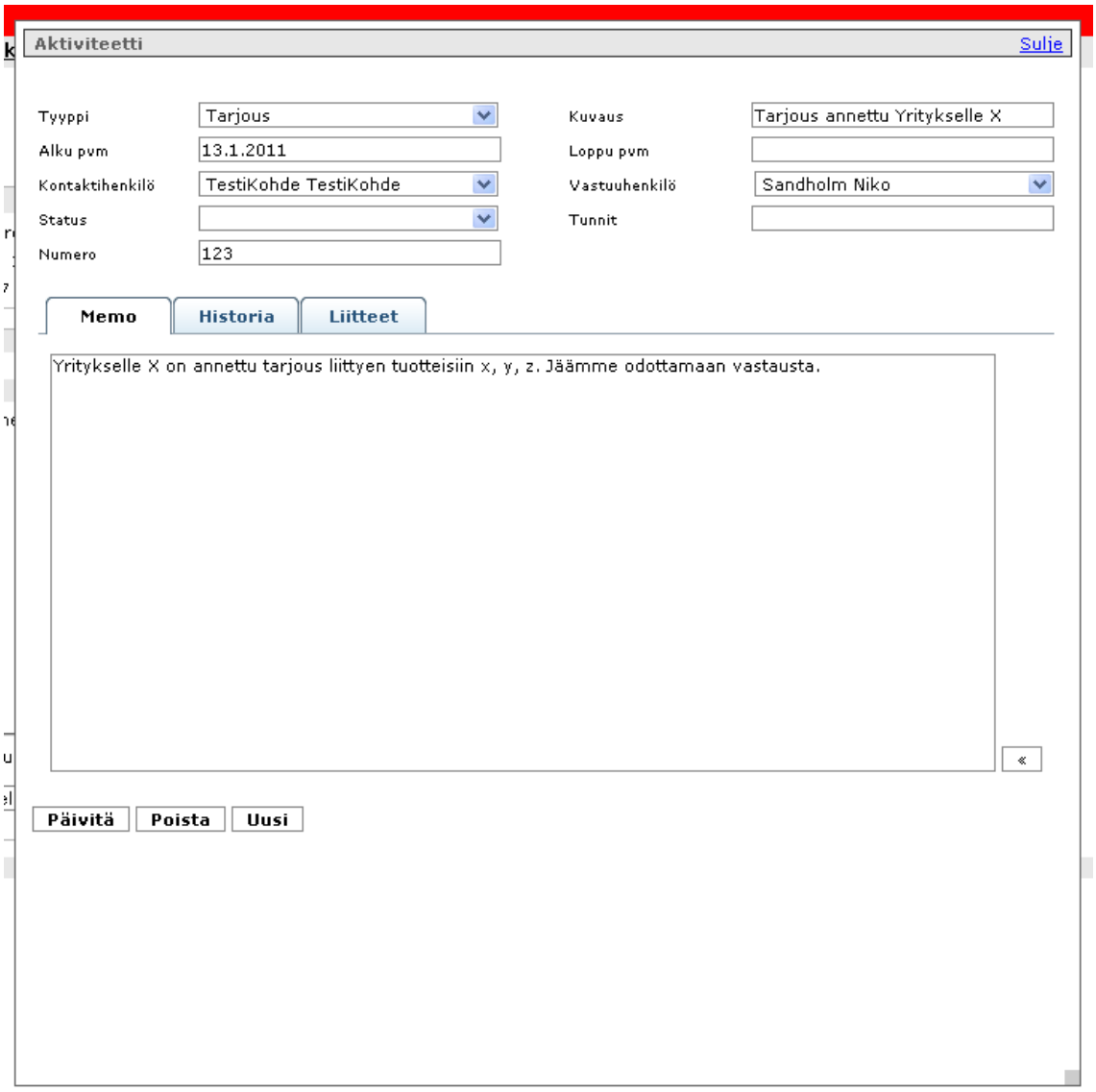


Kuva 2. Asiakkaaseen liittyvät aktiviteetti-ikkuna.

Asiakkaaseen liittyvät aktiviteetit aukeavat klikkaamalla oikealla olevasta navigointi paneelista Aktiviteetit linkkiä. Palvelimelta haetaan Ajax-kutsun avulla ainoastaan asiakkaaseen liittyvät aktiviteetit ja päivitetään ainoastaan taulukkorakenteella tehdyn sivun keskiosa, kun taas muu sivun sisältö jää ennalleen. Aktiviteettinäkymässä tietokannasta haetaan ja listataan asiakkaaseen liittyvät aktiviteetit. Ennen aktiviteetteja listattaessa näytetään kunkin aktiviteetin päiväys, tyyppi, status, vastuu henkilö ja kuvaus.

Sähköpostiaktiviteetti on tässä muodossaan uusi ominaisuus. Listausnäkymään lisättiin myös uusi sarake nimeltä liite, jonka alla on ilmaistu aktiviteettiin liittyvät liitteet tiedosto kuvakkeella ja liitteiden määrä numerolla sen perässä. Liitteiden hallinta -ominaisuus on tyypiltään tukiominaisuus, jonka avulla aktiviteettien käsittely helpottuu. Sähköpostiviestejä pilkottaessa viestistä saadaan prosessoitua tiettyyn asiakkaaseen liittyvät liitteet, tallennettua ne käyttäjän omaan tiedostokansioon, ja linkitettyä dynaamisesti tietokannan kautta kukin tiedosto siihen liittyvään aktiviteettiin.

Klikkaamalla aktiviteetin päiväystä, palvelu suorittaa Ajax-kutsun, joka hakee aktiviteettiin liittyvän tiedon tietokannasta ja näyttää sen ns. div-popup-ikkunassa (kuva 3). Popup on perinteisessä mielessä uusi selainikkuna, mutta myös sivustorakenteeseen upotettuja ikkunoita on mahdollista hyödyntää. Div-popup on toteutettu MooTools-JavaScript-kirjastoa hyödyntämällä, ja se on osa palvelun olemassa olevaa rakennetta. Div-popupin toiminnallisuutta ei ole muokattu uutta ko. toiminnallisuutta lisättäessä.

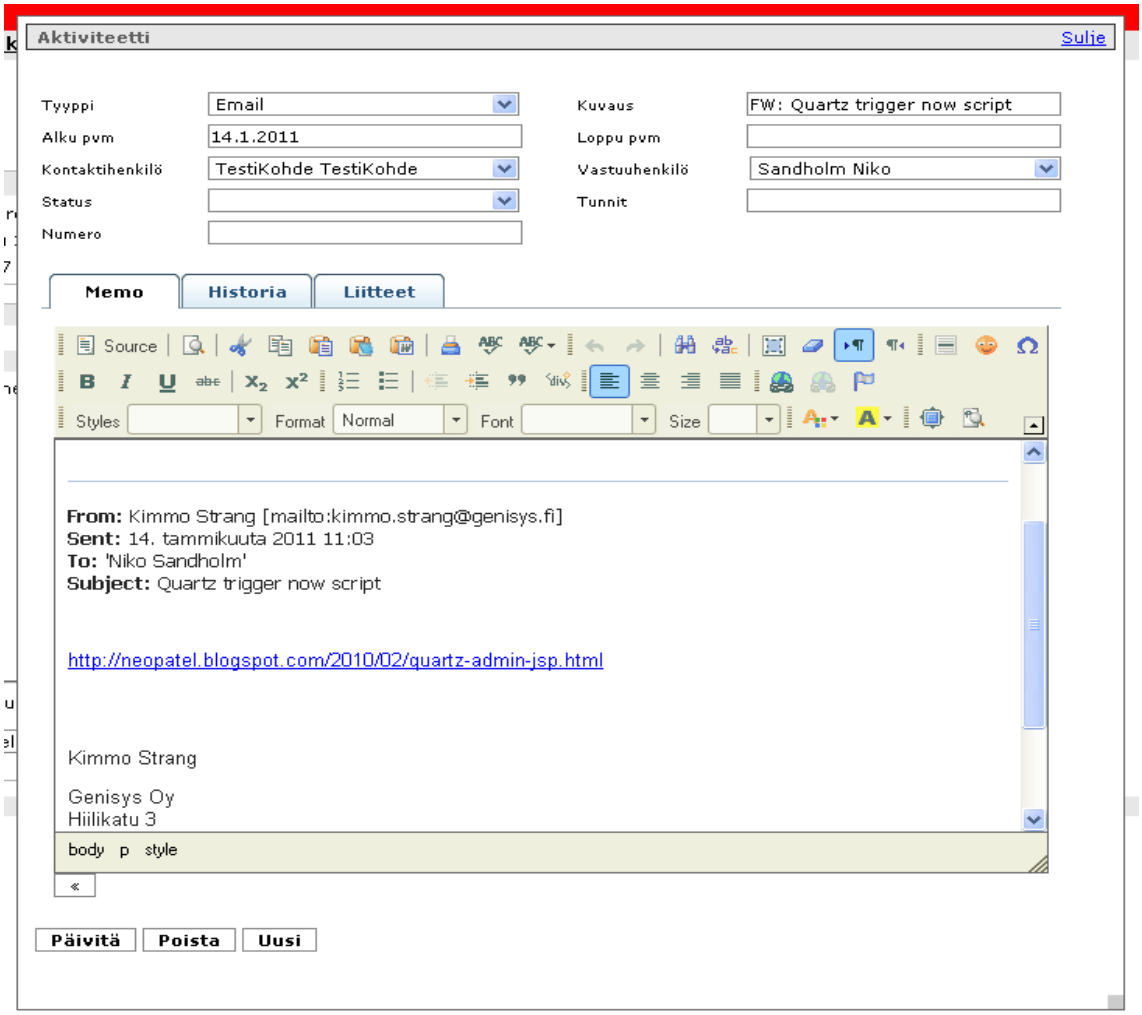


Kuva 3. Aktiviteetti-ikkuna, kun aktiviteettinä tarjous.

Aktiviteetti-ikkunaan tehtiin suurimmat muutokset. Ennestään aktiviteetti-ikkuna piti sisällään aktiviteetin tyypin eli minkälaisesta aktiviteetista on kyse, esim. tapaaminen, tarjous, sähköpostiviesti ym. ennalta määritellyt tyypit. Tyyppitiedot haetaan

dynaamisesti tietokannasta ja pudotusvalikko rakennetaan sen pohjalta. Klikkaamalla valikkoa kaikki tyypit saa näkyviin. Kuvaus, numero, loppumis päivämäärä ja tunnit ovat staattisia tekstikenttäelementtejä, joihin käyttäjä voi syöttää vastaavaa tietoa. Alkupäivämäärä on lomakkeen ainoa pakollinen tieto sitä tallennettaessa.

Ikkuna sijoittuu div-elementin sisään, joka edelleen aukeaa lomakepohjana, jonka avulla tiedot lähetetään eteenpäin. Lomakkeen rakenne pohjautuu pöytärakenteeseen, jossa on tietty määrä rivejä ja sarakkeita. Näin saadaan elementit asemoitua helposti ja hallitusti haluttuihin paikkoihin ja pysymään siellä, vaikka itse ikkunan koko muuttuisi. Kuvassa 4 näkyy aktiviteetti, joka on tyypiltää sähköpostiviesti. Ikkuna mukautuu käsiteltävän sisällön mukaan halutunlaiseksi.



Kuva 4. Aktiviteetti-ikkuna, kun aktiviteetin pohjana on sähköpostiviesti.

Kun käyttäjä valitsee alasvetovalikosta uuden arvon aktiviteetin tyypille, joka poikkeaa edellisestä, laukeaa JavaScript-onchange-tapahtuma. Tähän tapahtumaan on

kiinnitetty toggleContentDiv-funktio, joka suoritetaan aina, kun kyseinen tapahtuma ilmenee pudotusvalikkoelementissä. Valikon valinta välittyy eteenpäin arvioitavaksi funktion parametrina. Funktiossa suoritetaan valintalausekkeiden avulla yksinkertainen algoritmi, joka tulkitsee parametrin arvon ja tarkistaa vastaako arvo sähköpostin tyyppiä vai ei. Jos tyyppi on muu kuin sähköpostiviesti, editorille Memo-tabin tekstikentälle ei tehdä mitään. Jos tyyppi taas on sähköpostiviesti, tietyn valintalausekkeen ehdot täyttyvät, ja se laukaisee uuden CKEditor-instanssin kutsufunktion. CKEditor itsessään on JavaScript-pohjainen kevyt selaimessa toimiva verkkosivueditori. Sen on avoimen lähdekoodin ratkaisu ja saatavilla osoitteesta www.ckeditor.com.

Käyttäjä voi halutessaan muuttaa aktiviteettien tyyppiä yksinkertaisesti vaihtamalla tyyppivalikosta uuden tyypin ja painamalla päivitä. Näin ollen aktiviteetin uusi tyyppi tallentuu tietokantaan ja kun käyttäjä uudestaan avaa aktiviteetin listaus näkymästä, niin kyseisen Ajax-kutsun yhteydessä kutsutaan myös toggleContentDiv-funktiota, joka automaattisesti muuntaa näkymän vastaamaan aktiviteetin tyyppiä.

Jotta käyttöliittymään saatiin toiminnallisuutta, joka vastaa yllä mainittua kuvausta, lisättiin neljä uuttaa JavaScript-funktiota, joiden avulla CKEditor-instanssin esiintymistä hallinnoidaan. Funktiot ovat nimeltään toggleContentDiv, createCKEditor, saveCKEditorData ja deleteCKEditorOnLoad. Näiden lisäksi muutama koodirivi lisättiin ajaxPostForm-funktioon, joka on vastuussa div-popup-ikkunan tietojen lähettämisestä tietokannalle tallennettavaksi.

Kuten aiemmin jo mainittiin, toggleContentDiv-funktio sisältää valintalausekkeita, joiden avulla joko luodaan tai poistetaan editori instanssin aktiviteetti näkymästä. Se saa parametreina tyyppi id, joka on kullekin aktiviteetin tyypille määritelty yksilöllinen tunnus. Tyyppi on oleellisessa osassa valintalausekkeita, joiden avulla editori joko luodaan tai poistetaan. Se saa arvonaan myös tekstikentän, joka editorilla korvataan, ja myös tiedon siitä, onko näytettävä sähköpostiaktiviteetti tyyppiä HTML vai puhdasta tekstiä. Jos asiakas esimerkiksi on käynyt sähköpostiviestintää puhtaassa tekstimuodossa, pitää sähköpostit esittää tavanomaisessa tekstikentässä HTML-editorin sijaan, jotta tekstin muotoilut saadaan säilytettyä oikein.

Funktiot creakeCKEditor ja deleteCKeditorOnLoad ovat toggleContentDiv-funktion kutsumia apufunktioita, joiden avulla editori instanssi käytännössä luodaan ja tuhotaan.

Ne hyödyntävät CKEditorin oman JavaScript APIn metodeita toiminnassaan. Editoria luotaessa se alustetaan ensin tietyillä ominaisuuksilla, jotka säätelevät mm. työkalupalkin valikoita ja editorin ulkoasua.

Funktio saveCKEditorData on niin ikään apufunktio, jonka avulla varmistetaan käyttäjän syöttämän tiedon säilyvyys editori instanssissa. Käytännössä tämä tarkoittaa sitä, että jos käyttäjä on luomassa uutta aktiviteettia, ja luo editori instanssin div-popup-ikkunassa, pitää tiedot päivittää editorin instanssista myös alla olevaan teksti kenttään. Funktio on sidottu rivielementin on-mouse-over-tapahtumaan, joka on sijoitettu samalle taulukon riville missä Päivitä- ja Syötä-painikkeet sijaitsevat. Tapahtuma on onchange-tapahtuman lisäksi yksi monista selainten tukemista tapahtumista.

Kyseisten funktioiden ensisijainen tehtävä on tuoda haluttu toiminnallisuus käyttöliittymään, mutta myös tehdä palvelusta käyttäjäystävällisempi. Niiden avulla toiminnallisuus on tarkoitus tehdä helppokäyttöistä, ja saada näyttämään yksinkertaiselta, joka myös kestää käyttäjän tekemiä virheitä ja mahdollistaa palvelun suoraviivaisen toiminnan joka tilanteessa, mitä käyttäjä ikinä lomakkeella tekeekään.

Kaikkiin tilanteisiin ei ole palvelussa tarkoitus varautua, koska liiallinen kontrolli voi myös heikentää palvelun laatua. Tarpeettomat varmistukset ja käyttäjän hyväksynnän hakeminen tekee käyttäjänkin toiminnasta tehottomampaa, koska silloin palvelun funktionaalisuus häviäisi erilaisten turvapuskurien muodossa esiintyvien varmistuspyyntöjen alle.

3.4 Ylläpito

Kun pilvipalvelu on julkaistu, alkaa ohjelmiston mahdollinen jatkokehitys ja palvelun ylläpito taustatoimenpiteinä. Periaatteessa ohjelmallisiin virhetilanteisiin pyritään puuttumaan mahdollisimman ajoissa, mielellään heti ohjelmavirheiden ilmaannuttua ja tultua kehittäjien tietoisuuteen. Käytännössä katsottuna tämä on kuitenkin monimutkainen prosessi eikä virheitä päästä korjaamaan heti, kun ne ilmenevät käyttäjälle.

Vaikka palvelua olisi testattu intensiivisesti, kaikkia osatekijöitä ja niiden mahdollisia vaikutuksia ei välttämättä ole osattu ottaa huomioon riittävällä tarkkuudella. Näin ollen varsinaisen palvelukehityksen jälkeen voi ilmetä, että ohjaamakoodiin on jäänyt jonkin asteinen virhe ja se ilmenee käyttäjälle tavalla tai toisella. Riippuen virheen laadusta ja

sen ilmaantumispaikasta seurauksena voi olla, että ohjelmakoodi jatkaa toimintaansa virheestä huolimatta tai että ohjelman suoritus lakkaa kokonaan. Joissain tapauksissa ohjelmakoodissa voidaan vaikuttaa suoraan siihen, minkälaiset jälkiseuraamukset tulevat, kun virhe on tapahtunut. Tilanteesta riippuen voi olla mahdollista antaa ohjelman vain generoida virheilmoitus tapahtumasta ja sen jälkeen jatkaa toimintaa. Usein voi kuitenkin olla parempi, että ohjelma kaatuu, koska silloin virhe ilmenee heti, ja syytä on helpompi lähteä etsimään, kun tiedetään, mikä ohjelmakoodin osa petti ja millä parametreilla se tapahtui. Hiljaiset virheet voivat jäädä raportoimatta kokonaan, tai ne ilmoittavat itsestään ainoastaan lokiin ja jäävät helposti huomaamatta, jos lokeja ei analysoida.

Sähköpostikäsittelijän kannalta oleellisimmat virheet ilmenevät pääasiallisesti käyttöliittymässä, koska liiketoimintalogiikka toimii ainoastaan taustapalveluna. Käyttäjä huomaa sen olemassaolosta vain, jos sähköpostikäsittelijä on käsitellyt virheellisesti jonkin viestin tai jättänyt sen kokonaan käsittelemättä, minkä seurauksena se näkyy virheellisesti käyttöliittymässä tai ei näy siellä ollenkaan.

4 Johtopäätökset

4.1 Yhteenveto

Projektin pääosat ovat asiakkaan käymä sähköpostiviestintä eri asiakkaiden kanssa, tämän tiedon tallentaminen CRM-tietokantaan myöhempää analysointia varten ja tiedon näyttäminen asiakkaalle helpossa ja yksinkertaisessa muodossa. Kyseisen ohjelmistoprojektin päämääränä oli saavuttaa lopputuote, joka tyydyttää edellä määritellyn tarpeen.

Projekti toteutettiin ketterien kehitysmenetelmien periaatteita noudattaen. Mallia otettiin Scrum-menetelmästä, joka on hyvin projektiorientoitunut kehitysmenetelmä. Scrumissa määritellyt periaatteet toimivat ohjenuorana kehityksen etenemiselle ja projektin saattamisessa päätökseen mahdollisimman hallitusti ja suoraviivaisesti.

Käytettäväksi teknologiaksi valittiin Java EE -kehitysalusta Genisysin jo käyttäessä sitä ekstensiivisesti ja eksklusiivisesti muissakin pilvipalveluissaan oli luonnollista hyödyntää tätä tekniikkaa myös sähköpostinkäsittelijäprojektissa. Liiketoimintalogiikkaa toteutettiin puhtaasti Java-ohjelmointikielellä hyödyntäen Genisysin valmiiksi kehittämiä kirjastoja ja Java EE -alustan tarjoamia apukirjastoja. Erityisesti JavaMail API oli suuressa osassa liiketoimintalogiikan toteuttamista. Käyttöliittymä toteutettiin hyödyntäen Java EE -kehitysympäristön tarjoamia JavaServer Page -sivuja. Dynaamista sisältöä itse web-sivuille toteutettiin Java-kielellä sekä JavaScript- ja HTML DOM -teknologiaa hyödyntäen. Pyrkimyksenä oli tarjota käyttäjälle helppokäyttöinen palvelu miellyttävällä käyttökokemuksella.

Projekti aloitettiin toteuttamalla ensin liiketoimintalogiikka, jonka jälkeen siirryttiin käyttöliittymän rakentamiseen. Liiketoimintalogiikka perustuu sähköpostiviestin hakemiseen Genisysin sähköpostipalvelimelta, minkä jälkeen ne käsitellään viestikohtaisesti ja tallennetaan tietokantaan myöhempää tarkastelua varten. Viestien käsittelyn monimutkaisuus perustuu ennalta tuntemattomien viestin purkamisesta osiin ja niiden käsittelystä kullekin viestikomponentille ominaisella tavalla. Koska viestityyppejä on useita erilaisia, esim. liitteellisiä tai liitteettömiä, kuvia ja muotoilua sisältäviä viestejä, piti viestien käsittelyssä varautua useiden erilaisten viestien rakenteellisiin kokonaisuuksiin, jotka lopulta osa kerrallaan pitää purkaa helposti käsiteltäviksi komponenteiksi.

Käyttöliittymän toteuttaminen oli rakenteellisesti suoraviivaisempaa, mutta kuitenkin omalta toteutustavaltaan haastava kokonaisuus. Koska rakennetta toiminnallisuus tulisi olemaan osa GeniCRM-palvelua, ominaisuus tuli integroida olemassa olevaan CRM-palvelun käyttöliittymään. Sähköpostiviestejä käsiteltäisiin kuten muitakin CRM-palvelussa olevia aktiviteetteja, joten ne myös näytettäisiin samassa ikkunassa. Ikkunan rakenne muuttuisi riippuen siitä, onko kyseessä sähköpostiviesti vai jokin muu aktiviteetti. Kyseessä ollessa sähköpostiviesti ikkuna tuli korvata CKEditorilla, joka on kevyt web-pohjainen editori, joka laitettaisiin tavallisessa aktiviteetti-ikkunassa olevan Memo-kentän tilalle dynaamisesti. Useita eri variaatioita piti ottaa huomioon riippuen siitä, onko kyseessä uusi vai vanha viesti, ja tehdä päätöksiä, miten käyttöliittymä reagoi käyttäjän eri operaatioihin kussakin tilanteessa. Eri valinnoilla yritettiin luoda mahdollisimman looginen ja käyttäjäystävällinen ratkaisu, jotta käyttäjälle jäisi palvelusta miellyttävä käyttökokemus.

Tekniikkaa ja menetelmiä eivät tuottaneet liiemmin yllätyksiä. Java-kielen ollessa ennestään tuttu ei se aiheuttanut kehitystiimin parissa juurikaan ongelmia. JavaScript-kieli oli myös ennestään tuttu, kuten myös HTML DOM-malli, mutta tiimillä oli vähän käytännön kokemusta niiden käytöstä. Tämä tuotti joitakin haasteita ohjelmakoodia analysoitaessa virhetilanteiden ilmetessä. Java-kielen kääntäjä antaa aina jonkinlaisen virheilmoituksen, jos kielen kääntämisessä tietokoneelle ymmärrettävään muotoon tapahtui virhe, ja usein editori ennalta ehkäisee kriittisimpien virheiden synnyn. JavaScript-kielen kääntämisen ollessa täysin selaimen vastuulla se usein hiljentää tapahtuneet virheet, joten virheiden etsintä ja tulkinta jää täysin kehittäjän varaan. Ketterien menetelmien periaatteet kuitenkin avustivat kehitystä paljon, kun saatavan toiminnallisuuden purki tarpeeksi pieniin inkrementteihin ja iteroi riittävästi monesti niin ratkaisun ilmenee lopulta.

Projekti saatiin lopulta kuitenkin vietyä päätökseen ilman suurempia komplikaatioita. Erilaisten viestien moninaisten rakenteiden tunnistaminen ja käsittely tuottivat haasteita. Useimmat viestityypit saatiin sisällytettyä tuettuihin viestityyppeihin. Jos tulevaisuudessa ilmenee tarvetta laajentaa tuettujen viestityyppien kantaa, niin se toteutetaan tilannekohtaisesti ja jää jatkokehityksen varaan. Käyttöliittymän rakenteen muokkaaminen osoittautui verraten haastavaksi, ja CKEditor-instanssin integroiminen ja dynaaminen hallinta tuottivat erilaisia sisällöllisiä haasteita. Instanssin liittäminen Memo-viestikenttään ja siitä irrottaminen niin, että lomakkeeseen jo täytetyt tiedot

pysyvät tallessa, saatiin lopulta yksinkertainen ratkaisu muutaman iteraatiokierroksen jälkeen.

Lopulta uusi ominaisuus saatiin integroitua osaksi GeniCRM-palvelua, joka oli projektin päämäärä. Ohjelmistoprojektin lopputulos vastaa määrittelyvaiheessa palvelulle asetettuja vaatimuksia, joten projektin voidaan katsoa onnistuneen.

4.2 Jatkotoimenpiteet

Sähköpostien tuonti CRM-palveluun oli hyvin spesifioitu ratkaisu kyseiseen liiketoiminnalliseen ongelmaan. Sähköpostien prosessointilogiikkaa pyritään kehittämään tehokkaammaksi analysoimalla olemassa olevaa ohjelmakoodia, mutta ainakaan tällä hetkellä sitä ei nähdä yrityksen kannalta oleellisimpana asiana, johon olisi syytä sitoa resursseja. Käyttöliittymää pyritään kehittämään niin, että tarjoaa asiakkaille parhaan mahdollisen käyttökokemuksen pysyessään helppokäyttöisenä ja yksinkertaisena. Itse toiminnallisuutta ei ole syytä ruveta muokkaamaan nykyisten tietojen valossa.

Sähköposti toiminnallisuuden laajentaminen ja integroiminen yleisemmin voi olla mielekäs idea. Kun sähköposti toiminnallisuutta on onnistuneesti tuotu CRM-palveluun, mahdollisena jatkotoimenpiteenä palvelua ruvetaan kehitetään vastaamaan kevyttä sähköpostiohjelmaa tavalla tai toisella. Sähköpostiohjelman kautta asiakas voisi viestittää asiakkailleen suoraan CRM-palvelun kautta. Genisys tarjoaa jo nyt GeniMail-tyyppistä sähköpostimarkkinointipalvelua, joka tarjoaa valmiin rajapinnan sähköpostien lähettämiseen.

4.3 Projektin arviointi

Kokonaisuudessa projekti onnistui hyvin. Pilvipalvelun uusi ominaisuus saatiin toimimaan testeissä hyvin ja luotettavasti, ja nyt se on julkaistu yrityksen palvelimelle asiakkaiden käytettäväksi. Uusi ominaisuus ratkaisee kyseessä olevan liiketoimintaongelman yksinkertaisesti ja tehokkaasti, mikä oli projektin varsinainen päämäärä.

Ohjelmistoprojektina kokemus oli mielenkiintoinen ja avartava. Ollessani ensimmäistä kertaa mukana kaupallisessa ohjelmistoprojektissa en osannut hyödyntää ketterien menetelmien periaatteita yhtä selkeästi kuin olin olettanut, vaikka menetelmien oleelliset elementit, kuten iterointi ja inkrementointi, olivat selkeästi läsnä. Haastetta

tuottivat myös koodin tuottaminen ja samanaikainen auditointi, mitkä ovat usein ohjelmistoprojekteissa jaettu eri henkilöille.

Tuotettu ohjelmakoodi toimii hyvin. Sähköpostiviestien käsittely prosessi on jaoteltu eri funktioiden kesken selkeästi, jotta oleelliset tapahtumat ja niiden vaatima toiminnallisuus saadaan eriteltyä omiksi kokonaisuuksikseen, mikä helpottaa niiden hallittavuutta. Tämä helpottaa myös oleellisesti mahdollisten ohjelmointivirheiden paikannettavuutta ylläpidon näkökulmasta. Nuorempana ohjelmistokehittäjänä koodin kriittinen analysointi optimaalisen koodin, ei ole yhtä luonnollista minulle kuin kokeneemmalle kehittäjälle.

Käyttöliittymäsuunnittelun osuus jäi projektissa hieman häilyväksi. Itse aktiviteetti -ikkunan rakenteeseen ei paljonkaan voinut vaikuttaa, koska palvelun toiminnallisuuden eli sähköpostiviestin näyttämisen haluttiin tapahtuvan muiden aktiviteettien tapaan jo valmiiksi olemassa olevassa näkymässä. Tehtäväksi jäi muokata olemassa olevaa rakennetta niin, että sähköpostinäkymä saadaan integroitua aktiviteetti näkymään huomaamattomasti.

Lähteet

Distributed Multitiered Applications - The Java EE 5 Tutorial. 2010. Oracle. Verkkodokumentti. <http://download.oracle.com/javaee/5/tutorial/doc/bnaay.html>. Luettu 14.2.2011.

Document Object Model (DOM) Level 3 Core Specification. 2004. W3C. Verkkodokumentti. <http://www.w3.org/TR/DOM-Level-3-Core/>. Luettu 22.2.2011.

Genisys. 2010. Genisys Oy. Verkkodokumentti.<http://www.genisys.fi/>. Luettu 13.3.2011.

Haikala Ilkka, Märijärvi Jukka. 2004. Ohjelmistotuotanto. Helsinki: Talentum.

HTML 4.01 Specification. 1999. W3C. Verkkodokumentti. <http://www.w3.org/TR/html401/>. Luettu 21.2.2011.

Introduction to ISO 13407. 1999. European MultiMedia Usability Services. Verkkodokumentti. <http://www.ucc.ie/hfrg/emmus/methods/iso.html>. Luettu 13.3.2011.

Java: The Best Environment for Network-Based Applications. 2010. Oracle. Verkkodokumentti. <http://www.oracle.com/us/technologies/java/10045230-br-java-c17307-187867.pdf>. Luettu 3.3.2011.

Java EE Containers - The Java EE 5 Tutorial. 2010. Oracle. Verkkodokumentti. <http://download.oracle.com/javaee/5/tutorial/doc/bnabo.html> Luettu 26.2.2011.

JavaScript Guide. 2011. Mozilla Developer Network. Verkkodokumentti. <https://developer.mozilla.org/en/JavaScript/Guide>. Luettu 3.2.2011.

Ketterät käytännöt- Iteraatiot ja inkrementit. 2011. Sininen Meteoriitti Oy. Verkkodokumentti. < http://www.ketteratkaytannot.fi/fi-FI/Ketteryys/IteraatiotJaInkrementit/>. Luettu 2.1.2011.

Ketterät käytännöt- Scrum. 2011. Sininen Meteoriitti Oy. Verkkodokumentti. <http://www.ketteratkaytannot.fi/fi-FI/Menetelmat/Scrum/>. Luettu 2.1.2011.

Kuha Janne. 2008. Tehokas Java EE –sovellustuotanto. Jyväskylä: Docendo.

Principles behind the Agile Manifesto. 2001. Beck Kent, Beedle Mike, van Bennekum Arie, Cockburn Alistair, Cunningham Ward, Fowler Martin, Grenning James, Highsmith Jim, Hunt Andrew, Jeffries Ron, Kern Jon, Marick Brian, Martin Robert C., Mellor Steve, Schwaber Ken, Sutherland Jeff, Thomas Dave. Verkkodokumentti. <http://agilemanifesto.org/principles.html>. Luettu 3.2.2011.

Salo Immo. 2010. Cloud Computing – palvelut verkossa. Jyväskylä: Docendo.

Sinkkonen Irmeli, Nuutila Esko, Törmä Seppo. 2009. Helppokäyttöisen verkkopalvelun suunnittelu. Helsinki: Tietosanoma.

Scrum process. 2009. Lakeworks. Verkkodokumentti.
<http://commons.wikimedia.org/wiki/File:Scrum_process.svg>. Luettu 13.3.2011.

The Language List. 2011. Kinnersley Bill. Verkkodokumentti.
<http://people.ku.edu/~nkinners/LangList/Extras/langlist.htm>. Luettu 21.2.2011.

Sovelluslogiikan ohjelmakoodi

Erillinen liite, tarvittaessa arvioijan nähtävissä.